

ПРОГРАММИРОВАНИЕ
НА ЯЗЫКЕ
АССЕМБЛЕРА
ЕС ЭВМ

Hubes





ПРОГРАММИРОВАНИЕ
НА ЯЗЫКЕ
АССЕМБЛЕРА
ЕС ЭВМ



МОСКВА
«СТАТИСТИКА» 1976

З. С. БРИЧ, В. И. ВОЮШ, Г. С. ДЕГТЯРЕВА, Э. В. КОВАЛЕВИЧ

**П78 Программирование на языке Ассемблера ЕС ЭВМ.
М., «Статистика», 1976.**

296 с. с ил.

На обороте тит. л. авт.: З. С. Брич, В. И. Воюш, Г. С. Дегтярева, Э. В. Ковалевич.

В книге описывается система программирования на базе Ассемблера ЕС ЭВМ. Приводится полное описание языка Ассемблера, включая макросредства, и необходимые сведения о трансляторе Ассемблера в объеме, достаточном для его эксплуатации в рамках операционной системы ДОС ЕС. Изложение сопровождается примерами составления программы и макроопределений.

Книга рассчитана на специалистов, занимающихся программированием на ЕС ЭВМ. Она также может быть использована как учебное пособие по программированию на Ассемблере.

П 30502-151
008(01)-76 72-75

6Ф7.3

ПРЕДИСЛОВИЕ

Системы программирования на базе символических машинно-ориентированных языков программирования типа Ассемблер занимают важное место в программном обеспечении ЭВМ. Средства символического программирования, являясь обязательной частью большинства современных операционных систем, широко применяются программистами разных направлений. По сравнению с более развитыми системами программирования символическое программирование обладает рядом преимуществ, когда необходимо получить более качественную программу или требуется максимально использовать технические возможности ЭВМ, а также когда универсальные или проблемно-ориентированные языки программирования не дают удовлетворительного результата. Об этом хорошо знают системные программисты, для которых Ассемблер является основным инструментом. Не случайно поэтому разработка системного программного обеспечения ЭВМ начинается, как правило, с реализации Ассемблера. Широко распространено использование символических языков в прикладном программировании. Как и другие языки программирования, Ассемблеры будут совершенствоваться, например, в направлении дальнейшего развития идей «макро». Более полная реализация этих идей может еще более упрочить позиции символического машинно-ориентированного программирования.

Несмотря на машинную ориентацию символических языков, Ассемблеры обладают многими, выработанными многолетней практикой программирования, общими чертами, часть которых является следствием логической близости тех вычислительных систем, на которые они ориентированы. Общность Ассемблеров наиболее характерна для ЭВМ третьего поколения, к которому принадлежат целые семейства машин IBM (США), Siemens (ФРГ), ICL (Англия), а также разработанная странами — участницами СЭВ Единая Система электронных вычислительных машин (ЕС ЭВМ).

Предлагаемая книга посвящена Ассемблеру машин ЕС ЭВМ. Этот машинно-ориентированный язык программирования является типичным представителем Ассемблеров ЭВМ третьего поколения и в программное обеспечение машин ЕС ЭВМ входит как составная часть операционных систем ОС ЕС и ДОС ЕС. Изложе-

ние Ассемблера дается в рамках операционной системы ДОС ЕС. Программирование на Ассемблере ОС ЕС имеет отличия, вызванные, в основном, особенностями концепций, положенных в основу самой операционной системы ОС ЕС, а также за счет некоторого расширения языка Ассемблера ОС ЕС. Описание этого расширения приведено в приложении 1.

Книга содержит достаточно полное описание языка и необходимые краткие сведения о трансляторе Ассемблера ДОС ЕС с тем, чтобы она могла оказать помощь при практическом и учебном программировании. Для разъяснения отдельных элементов или функций Ассемблера приводятся примеры, в том числе примеры коротких программ и макроопределений.

Книга рекомендуется пользователям ЕС ЭВМ. Наряду с другими пособиями по операционным системам ЕС ЭВМ она может быть также использована для учебных целей.

ОБЩИЕ СВЕДЕНИЯ О ДАННЫХ, КОМАНДАХ И АССЕМБЛЕРЕ ЕС ЭВМ

1.1. ШЕСТНАДЦАТЕРИЧНАЯ СИСТЕМА СЧИСЛЕНИЯ

В памяти ЕС ЭВМ, как и большинства других ЭВМ, информация представляется в двоичной системе счисления, но при программировании для удобства записи применяется шестнадцатеричная система счисления. Для изображения чисел в шестнадцатеричной системе счисления требуется 16 цифр. Десятичная система счисления предоставляет только 10 цифр (0—9), для изображения шести остальных цифр требуется 6 дополнительных знаков. Для этой цели принято использовать буквы А, В, С, D, E, F, хотя можно было бы использовать любые другие шесть знаков. Каждая шестнадцатеричная цифра представляется четырьмя двоичными цифрами. Двоичные эквиваленты шестнадцатеричных цифр следующие:

0 — 0000	4 — 0100	8 — 1000	C — 1100
1 — 0001	5 — 0101	9 — 1001	D — 1101
2 — 0010	6 — 0110	A — 1010	E — 1110
3 — 0011	7 — 0111	B — 1011	F — 1111

Двоичное число можно легко преобразовать в шестнадцатеричное. Для этого нужно выделить в двоичном числе группы по четыре двоичных цифры, причем для целой части выделение следует выполнять справа налево, а для дробной — слева направо. После этого нужно заменить каждую группу двоичных цифр шестнадцатеричной цифрой. Если самая левая группа целой части неполная, то она дополняется нулями. Аналогично поступают и с самой правой группой дробной части. Например:

$$\begin{aligned}
 111110011011010011 &= 0011/1110/0110/1101/0011 = \\
 &= 3 \quad E \quad 6 \quad D \quad 3 = \\
 &= 3E6D3 \text{ (в шестнадцатеричной системе)}.
 \end{aligned}$$

Для преобразования шестнадцатеричных чисел в двоичные необходимо заменить соответствующими группами из четырех двоичных цифр каждую шестнадцатеричную цифру, и все незначащие (стоящие перед самой первой единицей в числе) нули опустить. Например:

$$6C4 = 0110/1100/0100 = 11011000100 \text{ (в двоичной системе)}.$$

Методы перевода чисел из одной системы счисления в другую часто рассматриваются в литературе по вычислительной технике¹.

Арифметические действия над числами в любой системе счисления выполняются по тем же правилам (перенос в старший разряд и т. п.), что и над числами в десятичной системе счисления. Например, при выполнении шестнадцатеричного сложения, если сумма двух цифр превышает F (максимальную шестнадцатеричную цифру), происходит перенос единицы в старший разряд:

$$\begin{array}{r}
 \begin{array}{ccc}
 & 1 & 1 \\
 & \swarrow & \nwarrow \\
 + & 6 & A & E \\
 & 1 & F & A \\
 \hline
 & 8 & A & 8
 \end{array}
 \end{array}$$

1.2. ФОРМАТЫ ДАННЫХ И КОМАНД В ЕС ЭВМ

В ЕС ЭВМ могут обрабатываться данные нескольких форматов².

Восьмиразрядная единица информации, называемая байтом, является основой построения данных всех форматов. Байты могут обрабатываться каждый отдельно или полями, состоящими из нескольких байт. Основными обрабатываемыми единицами машинных команд являются:

- один байт;
- полуслово, представляющее собой группу из двух последовательно расположенных байт;
- слово, являющееся группой из четырех последовательно расположенных байт;
- двойное слово, состоящее из двух последовательно расположенных слов;
- группа байт, число которых не более 256.

В соответствии с данными, которые обрабатываются в ЕС ЭВМ, в системе команд можно выделить группы команд, выполняющие следующие операции: операции с фиксированной точкой, операции с плавающей точкой, операции над десятичными числами и логические операции. Команды каждой группы подробно рассматриваются в главе 3. Под операцией здесь понимается конкретное действие, выполняемое над данными. Команда — это указание, какую именно операцию и над какими данными должна выполнить вычислительная система.

В арифметических операциях с фиксированной точкой операндами являются числа с фиксированной точкой, двоичные

¹ Савинков В. М. Программирование для ЭЦВМ «МИНСК-32». М., «Статистика», 1972.

Ледли Р. Программирование и использование вычислительных машин. М., «Мир», 1966.

² Джермейн К. Программирование на IBM/360. М., «Мир», 1971.

Вычислительная система IBM/360. Принципы работы. М., «Советское радио», 1969.

целые числа со знаком. Название «число с фиксированной точкой» употребляется потому, что машина интерпретирует это число как двоичное число, у которого точка, отделяющая целую часть от дробной, находится справа от последней значащей цифры. Числа с фиксированной точкой в памяти машины имеют формат фиксированной длины: слово или полуслово (рис. 1).

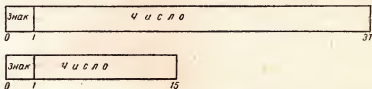


Рис. 1. Формат чисел с фиксированной точкой

В этом формате один бит отводится под знак, а последующие 31 или 15 бит образуют поле целой части. Положительные числа представляются в прямом коде со знаковым разрядом, равным нулю. Отрицательные числа представляются в дополнительном коде со знаковым разрядом, равным единице. Дополнительный код числа получается инвертированием каждого разряда числа с последующим прибавлением единицы к младшему разряду. В операциях с фиксированной точкой отделение целой части числа от дробной путем установления точки в нужном месте возлагается на программиста.

Чтобы освободить программиста от обязанности следить за положением точки во время вычислений, в ЕС ЭВМ обеспечивается выполнение операций с плавающей точкой. В операциях с плавающей точкой операндами являются числа с плавающей точкой, которые в памяти машины могут представляться в двух форматах фиксированной длины: коротком (число длиной в слово) и длинном (число длиной в двойное слово). Эти форматы отличаются друг от друга только длиной дробной части (рис. 2).

Число с плавающей точкой состоит из порядка и мантиссы (дробной части). Величина этого числа равна произведению мантиссы на число 16 (основание системы счисления), возведенное в степень, равную порядку. Порядок числа с плавающей точкой в памяти машины представляется не истинной величиной: он увели-

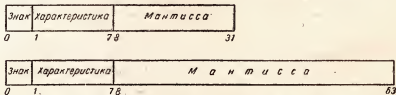


Рис. 2. Формат чисел с плавающей точкой

чен на 64 и называется характеристикой. Характеристика — всегда положительное число. Мантисса — шестнадцатеричное число, у которого точка находится слева от самой старшей шестнадцатеричной цифры. В памяти машины каждая шестнадцатеричная цифра представляется комбинацией двоичных нулей и единиц (двоичным кодом). Нулевой разряд в любом формате (коротком и длинном) отводится для знака числа. Следующие семь разрядов заняты характеристикой. Остальные разряды отводятся под мантиссу. Мантисса может состоять из шести либо из четырнадцати шестнадцатеричных цифр. Использование коротких операндов, обеспечивающих точность до семи десятичных знаков, позволяет размещать в памяти максимальное количество операндов и сокращает время выполнения программы. Длинные операнды используются в тех случаях, когда необходимо проводить вычисления с повышенной точностью.

В операциях десятичной арифметики операндами являются десятичные числа. Каждая цифра десятичного числа представляется четырехразрядной комбинацией двоичных нулей и единиц. Десятичные числа 0—9 выражаются в двоичном представлении четырехразрядными кодами от 0000 до 1001 соответственно. Для знака плюс выбрано двоичное представление 1100, для знака минус — 1101. Операнды могут иметь переменную длину до 16 байт (31 десятичная цифра плюс знак). Они могут быть представлены в упакованном формате или в формате с зоной (рис. 3). В упаков-

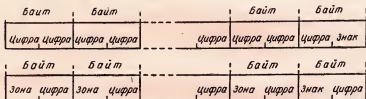


Рис. 3. Формат десятичных чисел

ванном формате две десятичные цифры расположены рядом в одном байте. Исключение составляет самый правый байт, в котором справа от цифры помещается знак. В формате с зоной младшие четыре разряда каждого байта заняты десятичной цифрой, старшие четыре разряда отводятся под зону. Зона представляет собой двоичный код 1111. В самом правом байте числа с зоной место зоны занято знаком числа.

Логические операции (например, пересылка знаков) используют логическую информацию, которая может иметь фиксированную и переменную длину. При использовании операндов фиксированной длины логическая информация может быть представлена одним, четырьмя или восемью байтами. Длина операндов переменной длины может достигать 256 байт.

Обычно логической информацией является текст, состоящий из букв, цифр и специальных знаков. Для представления такой информации в памяти ЕС ЭВМ каждому символу отводится один байт, т. е. 8 двоичных разрядов. Логические команды могут выполнять действия над любыми восьмиразрядными кодами. Однако внешние устройства Единой системы ЭВМ используют в основном Двоичный Код Обмена Информации (ДКОИ), в котором не всем возможным 256 восьмиразрядным кодам соответствуют графические символы. Для того чтобы представить все возможные комбинации бит кода ДКОИ, можно использовать их шестнадцатеричное представление, в котором один графический символ соответствует четырехбитовому коду. Следовательно, для обозначения байта достаточно двух графических символов — двух шестнадцатеричных цифр.

Таким образом, система команд ЕС ЭВМ позволяет обрабатывать данные разных типов. Для адресации этих данных необходимо учитывать следующие факторы:

байты памяти ЕС ЭВМ нумеруются последовательно, начиная с 0;

каждый номер считается адресом соответствующего байта;

при адресации группы байт указывается адрес самого левого байта.

Если команда использует операнд переменной длины, то он может начинаться с любого байта, в команде при этом должна быть указана длина данного. Операнды фиксированной длины должны размещаться в основной памяти, начиная с целочисленной границы для данной группы байт. Граница называется целочисленной для группы байт, если адрес этой группы байт кратен количеству байт в группе. Например, слово, состоящее из четырех байт, должно располагаться в памяти так, чтобы его адрес был кратен четырем.

Система команд ЕС ЭВМ включает в себя команды пяти форматов. Каждая команда определяет выполняемую операцию и данные, над которыми она выполняется. Операнды команд могут быть расположены в регистрах, в основной памяти или могут быть частью самой команды. В зависимости от расположения операндов команды имеют разную длину и время выполнения. Команды длиной в два полуслова содержат один адрес основной памяти, в командах длиной в три полуслова указываются два адреса основной памяти.

Пять форматов команд обозначаются следующими форматными кодами: RR, RX, RS, SI и SS. Форматные коды указывают место расположения операндов: RR обозначает операцию типа регистр — регистр; RX — операцию типа регистр — память, при этом адрес памяти индексируется; RS — операцию типа регистр — память без индексации; SI — операцию, когда один операнд находится в памяти, а другой представляет собой непосредственный операнд, т. е. присутствует в самой команде; SS — операцию типа

память — память. Все команды должны размещаться в памяти, начиная с целочисленной границы полуслова.

На рис. 4 показаны пять основных форматов команд. В каждом формате первое полуслово состоит из двух частей. Первый байт содержит код операции. Второй байт (два четырехразрядных поля или одно восьмиразрядное поле) может содержать следующую информацию:

- четырехразрядные номера регистров, в которых содержатся операнды (R1, R2 или R3);
- четырехразрядный номер регистра индекса (X1);
- четырехразрядную маску;
- четырехразрядный код длины операнда (L1 или L2);
- восьмиразрядный код длины операнда (L);
- восьмиразрядный код, представляющий непосредственный операнд (I2).

В некоторых командах четырехбитовое поле второго байта или весь второй байт в первом полуслове игнорируется.

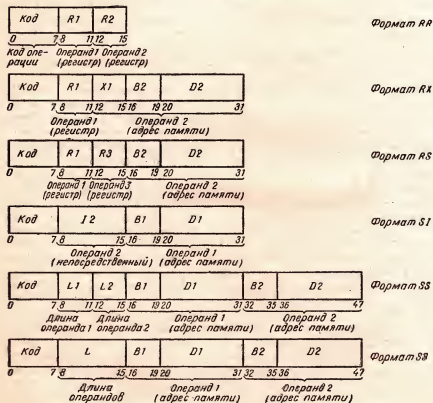


Рис. 4. Формат машинных команд

Во всех командах второе и третье полуслова всегда имеют один и тот же формат: четырехразрядный номер регистра базы (B1 или B2) и следующее за ним 12-разрядное смещение (D1 или D2).

1.3. СИМВОЛИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Аналогично большинству вычислительных машин в ЕС ЭВМ используется форма двоичного представления программ. Все программы, предназначенные для выполнения на ЕС ЭВМ, состоят из команд и данных, которые представляются двоичными числами. Создание программы заключается в записи набора двоичных чисел. Если такие числа строит и записывает программист, то это значит, что он создает программу на машинном языке. При записи программ на машинном языке обычно двоичное представление не используется, а всевозможные двоичные комбинации представляются шестнадцатеричными цифрами. Например, следующая последовательность шестнадцатеричных цифр представляет собой команду сложения с фиксированной точкой формата RX, которая осуществляет сложение слова основной памяти с содержимым общего регистра 10: 5 AA020AA. Эта команда написана на машинном языке шестнадцатеричными цифрами.

Писать команды на машинном языке для программиста сложно и утомительно. Гораздо проще записывать команды не в виде цифр, а как предложения текста. Например, вместо того чтобы написать 5AA020AA (сложение слова основной памяти с содержимым регистра 10), проще записать A 10, FIELD A, предположив, что буква A обозначает сложение (от английского слова ADD — сложить), 10 — номер регистра, FIELD A — название слова основной памяти. Такое программирование называется программированием на символическом языке или символическим программированием.

Использование символического языка облегчает написание программ, так как значительно легче и удобнее записывать адреса команды в виде текста, а не цифр. Но символическую программу невозможно сразу выполнить на ЭВМ, потому что ЭВМ может воспринимать только двоичные цифры. Поэтому для каждой команды, записанной на символическом языке, нужно создать команду на машинном языке (двоичное число). Этот процесс перевода программы с символического языка на машинный называется трансляцией. Выполнять трансляцию может любой, кто знаком с грамматическими правилами символического языка. Эта работа однообразная и трудоемкая, поэтому выполнение ее целесообразно возложить на ЭВМ. Можно создать программу, которая обрабатывала бы символические команды как входные данные и транслировала бы их в машинный эквивалент в соответствии с правилами символического языка. Один раз написав такую программу, ее можно неоднократно использовать для трансляции

множества символических программ на машинный язык. Программы, выполняющие трансляцию с символических языков программирования на машинный язык, называются трансляторами. Входной информацией для транслятора является символическая программа, написанная на соответствующем транслятору символическом языке. Выходной информацией после выполнения трансляции является программа на машинном языке, которая после обработки специальной программой (Редактором) непосредственно может выполняться на машине.

Символические языки программирования относятся к машинно-ориентированным языкам программирования. Машинно-ориентированным языком программирования для ЕС ЭВМ является язык Ассемблера.

1.4. КРАТКАЯ ХАРАКТЕРИСТИКА АССЕМБЛЕРА

1.4.1. Назначение Ассемблера

Собственно Ассемблер состоит из языка Ассемблера и трансляторов Ассемблера.

Язык Ассемблера, являясь символическим языком программирования, позволяет непосредственно использовать систему команд ЕС ЭВМ и предоставляет потребителю ряд удобств для кодирования программ на машинном языке.

Основные преимущества языка Ассемблера перед машинным языком:

- символическая адресация элементов программы;
- неявное представление в командах длин операндов;
- разнообразные способы представления данных;
- использование данных в операндах команд;
- наличие средств деления программы на части и связи между отдельно транслируемыми частями программы;
- наличие средств, позволяющих изменять порядок и содержание исходных операторов.

Язык Ассемблера может быть использован для написания более эффективных программ по сравнению с программами, написанными на других языках программирования. Между языками Ассемблера операционных систем ОС и ДОС существуют незначительные отличия, которые приведены в приложении 1.

Транслятор Ассемблера переводит программу, написанную на языке Ассемблера и называемую исходной программой или исходным модулем, в программу на машинном языке, называемую объектным модулем.

В ДОС ЕС существуют два транслятора Ассемблера: транслятор Ассемблера *E* и транслятор Ассемблера *F*. Различие между ними состоит в объеме памяти, необходимом для работы каждого транслятора. Входные языки для каждого из трансляторов отли-

чаются только некоторыми количественными характеристиками, которые при описании возможностей языка будут отмечены. Эти отличия также приведены в приложении 1.

1.4.2. Основные положения языка Ассемблера

Исходная программа на языке Ассемблера записывается в виде отдельных символических операторов. В языке Ассемблера используются следующие типы операторов: машинные команды, команды Ассемблера, макрокоманды, команды генерации, комментарии.

Операторы машинных команд представляют собой символическую форму записи обычных машинных команд ЕС ЭВМ. Набор операторов машинных команд определяется системой команд ЕС ЭВМ (приведен в приложении 2).

Операторы команд Ассемблера определяют действия транслятора при переводе исходной программы на машинный язык. Команды Ассемблера в свою очередь разделяются на команды определения, команды секционирования и соединения, команды управления.

Команды определения предоставляют программисту различные способы определения констант и областей памяти, присваивают значения символическим именам, определяют регистры базы. Команды секционирования и соединения позволяют делить программу на части, устанавливать связи между программными частями, которые будут транслироваться отдельно. Команды управления позволяют изменять содержание результатов трансляции, управлять вводом исходного модуля.

Операторы макрокоманд используются для обращения к макроопределениям (ранее подготовленным наборам операторов на языке Ассемблера).

Операторы команд генерации вместе с макрокомандами составляют макросредства языка Ассемблера. С помощью команд генерации можно менять последовательность генерации машинных команд и команд Ассемблера, а также их содержание.

Операторы комментариев поясняют программу на языке Ассемблера и никакого влияния на содержание создаваемого объектного модуля (программы на машинном языке) не оказывают.

Каждый символический оператор языка Ассемблера сообщает транслятору Ассемблера о необходимости построить команду на машинном языке, константу или выполнить во время трансляции какое-то действие.

Символические операторы записываются на специальном бланке кодирования, показанном на рис. 5. Для записи символического оператора на бланке кодирования отводится одна строка, которая может содержать четыре поля: поле названия, поле операции, поле операндов и поле комментариев. Поле названия используется для присвоения символических названий областям памяти, командам или данным, к которым обращаются в программе. На-

[illegible]

Рис. 5. Бланк кодирования

звания указываются в виде символических имен. Для того чтобы лучше ориентироваться при чтении символической программы, желательно, чтобы символические имена отражали смысл элементов, которые они именуют.

Чтобы показать, что представляет собой символический оператор, будет ли он машинной командой или другим оператором, используется поле операции. Символическое обозначение необходимой операции помещается в поле операции и называется мнемоническим кодом операции.

В поле операндов записываются операнды оператора. Здесь указываются операнды, над которыми должны производиться какие-либо операции.

Поле комментариев можно использовать для записи комментариев, поясняющих смысл и назначение записываемых операторов.

Мнемонические коды операций, содержание и формат записи поля названия и поля операндов всех операторов Ассемблера приведены в приложении 2.

1.4.3. Трансляция

Исходный модуль (программа на языке Ассемблера), записанный на бланке кодирования, перфорируется на перфокарты. Одна строка бланка кодирования перфорируется на одну перфокарту. Перфокарты составляют колоду исходной программы, которая представляет собой входную информацию, обрабатываемую транслятором Ассемблера. Транслятор Ассемблера считывает карты исходной колоды и переводит операторы исходной программы на машинный язык, представляя программу на машинном языке в формате объектного модуля. Дальнейшая обработка и подготовка программы к выполнению осуществляется компонентом ДОС ЕС Редактором.

Трансляция с языка Ассемблера выполняется в два этапа. На первом этапе осуществляются ввод исходной программы, включение макроопределений, указанных макрокомандами исходной программы, и копируемых текстов (копируемый текст — это текст, составленный из операторов на языке Ассемблера, который включается в текст исходной программы из библиотеки исходных модулей с помощью оператора Ассемблера `COPY`). Далее текст программы, расширенный макроопределениями и копируемыми текстами, обрабатывается согласно указаниям команд генерации; определяются порядок и содержание исходных операторов, которые будут обрабатываться на втором этапе.

На втором этапе мнемонические коды операций заменяются машинными; данные, записанные на языке Ассемблера, переводятся в машинный формат. Для рабочих областей резервируются области памяти, исходным операторам присваиваются значения адресов памяти. Для правильного присваивания адресов памяти символическим именам транслятор Ассемблера каждому оператору присваивает значение счетчика адреса. Для первого оператора

транслятор задает счетчику адреса некоторое первоначальное значение, затем счетчик адреса корректируется для каждого обработанного символического оператора. Адреса памяти в машинных командах представляются в виде регистра базы и смещения, как этого требует их машинное представление.

В конце второго этапа выдаются результаты трансляции. Результатом трансляции являются объектный модуль, представляющий собой программу на машинном языке, и распечатка результатов трансляции. Объектный модуль перфорируется на перфокарты, выводится на магнитную ленту и диск. Распечатка результатов трансляции содержит тексты объектного и исходного модулей и некоторые другие сведения, необходимые для анализа программы, например сообщения о синтаксических ошибках в исходной программе.

ЭЛЕМЕНТЫ ЯЗЫКА АССЕМБЛЕРА

2.1. ТЕРМЫ И ВЫРАЖЕНИЯ

В языке Ассемблера каждый операнд машинной команды или команды Ассемблера записывается как выражение, которое при трансляции получает значение. Выражение состоит из одного термина или арифметической комбинации термов.

Терм представляет собой некоторое значение, которое может присваиваться транслятором Ассемблера (символическое имя, ссылка на характеристику длины, значение счетчика адреса) либо может определяться самим термом (самоопределенный терм, литерал). В языке Ассемблера допускается пять видов термов: самоопределенный терм, символическое имя, значение счетчика адреса, ссылка на характеристику длины, литерал.

Для каждого термина определяются характеристика длины и признак переместимости. Характеристика длины для тех термов, которые определяют область памяти, равна количеству байт в этой области памяти. Для остальных термов характеристика длины присваивается по правилам, которые описаны ниже для каждого термина.

В зависимости от того, изменяется значение термина или остается постоянным при перемещении программы, термины разделяются на переместимые и абсолютные. Перемещение программы — это загрузка объектного модуля в область основной памяти, отличную от той, которая была первоначально назначена транслятором Ассемблера. Терм является абсолютным, если его значение не изменяется при перемещении программы. Если значение термина изменяется при перемещении программы, он является переместимым.

2.1.1. Самоопределенные термины

Самоопределенный терм — это терм, значение которого выражено в нем самом. Существуют четыре типа самоопределенных термов: десятичный, шестнадцатеричный, двоичный, знаковый.

Самоопределенные термины являются средством записи чисел в десятичном, шестнадцатеричном, двоичном или знаковом представлении.

Десятичный самоопределенный терм — это десятичное число без знака, записанное в виде последовательности десятичных цифр. В старших разрядах десятичного термина могут записываться незначащие нули. Десятичный самоопределенный терм не может

содержать больше восьми десятичных цифр. Значение десятичного термина не должно быть больше 16777215 ($2^{24}-1$). Десятичный терм во время трансляции переводится транслятором Ассемблера в свой двоичный эквивалент.

Шестнадцатеричный самоопределенный терм — это шестнадцатеричное число без знака, записанное в виде последовательности шестнадцатеричных цифр. Шестнадцатеричными цифрами являются знаки 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Цифры должны быть заключены в апострофы, первому апострофу должна предшествовать буква X, например, X'C49'. Максимальное значение шестнадцатеричного термина — X'FFFFFF'. Каждая шестнадцатеричная цифра шестнадцатеричного термина транслируется в свой двоичный эквивалент.

Двоичный самоопределенный терм — это последовательность нулей и единиц без знака, заключенная в апострофы, первому апострофу должна предшествовать буква B, например, B'10001101'. Двоичный терм может иметь до 24 двоичных цифр.

Знаковый самоопределенный терм — это последовательность знаков, заключенная в апострофы, первому апострофу должна предшествовать буква C. Знаковый самоопределенный терм может содержать от одного до трех знаков. В знаковом самоопределенном терме могут быть использованы все знаки кода ДКОИ. Так как апостроф и знак & в языке Ассемблера используются как синтаксические знаки, то необходимо придерживаться следующего правила использования этих знаков в знаковом терме: для каждого апострофа или знака &, который нужно использовать в знаковом самоопределенном терме, записываются подряд два апострофа или два знака &. Например, знаковое значение A'# должно быть записано как C'A"##'; совокупность знаков: апостроф, пробел и опять апостроф — должна быть записана как C'"'". Каждый знак из последовательности знакового термина транслируется в его восьмиразрядный двоичный эквивалент. Два знака & или два апострофа, используемые для представления апострофа или знака & внутри последовательности, транслируются в один апостроф или знак &.

Примеры самоопределенных термов:

X'100'
147
C'ABC'
C'~'
B'111110000'

Самоопределенные термы являются абсолютными терминами, так как значения, которые они представляют, не изменяются при перемещении программ. Характеристика длины самоопределенных термов всегда равна 1.

Самоопределенные термы можно использовать для определения таких элементов машинных команд, как непосредственный операнд, маска, регистры, смещение при указании адреса памяти. Обычно выбирается тот тип термина, с помощью которого легче за-

писывается элемент команды. Например, чтобы сложить содержимое общего регистра 14 с содержимым общего регистра 11, можно написать следующую команду:

Название	Операция	Операнды
	AR	X'B',X'E'

Но предпочтительнее записать эту команду, используя для обозначения регистров десятичные термы:

Название	Операция	Операнды
	AR	11,14

Шестнадцатеричные или двоичные самоопределенные термы часто применяются тогда, когда необходимо использовать битовую структуру данных. Например, необходимо установить все биты байта в единицу. Это можно сделать с помощью следующей команды:

Название	Операция	Операнды
	MVI	BYTEA,X'FF'

В байт с именем BYTEA пересылается набор из восьми единиц, который записан в виде шестнадцатеричного терма.

Предположим, необходимо, чтобы байт памяти содержал двоичное число¹ 11000011. Это можно сделать с помощью следующей команды:

Название	Операция	Операнды
	MVI	BYTEA,B'11000011'

С помощью двоичного терма можно легко записать в команде двоичное число B'11000011'. Следующий пример иллюстрирует использование двоичного терма в качестве маски в команде TM (ПРОВЕРИТЬ ПО МАСКЕ). Содержимое байта с именем GAMMA проверяется двоичным числом, представленным двоичным термом.

¹ На бланках кодирования цифра ноль перечеркивается (Ø) для того, чтобы ее можно было отличить от буквы O.

Название	Операция	Операнды
ALPHA	TM	GAMMA,B'1Ø1Ø11Ø'

Рассмотрим также пример использования знакового самоопределенного терма. Предположим, нужно проверить, содержится ли в байте с именем GARDIN знак «*». Эту проверку можно осуществить следующей командой:

Название	Операция	Операнды
	CLI	GARDIN,C'*

В зависимости от того, для каких целей используются самоопределенные термы, на значения термов накладываются соответствующие ограничения. Например, десятичный терм, используемый для обозначения общего регистра, должен принимать значения только от 0 до 15 включительно.

2.1.2. Символическое имя

Возможность использовать символические имена — одна из наиболее важных особенностей символического программирования. Символическое имя — это знак или комбинация знаков, используемая для обозначения адресов или других величин. Символические имена используются в поле названия и в поле операндов, предоставляя программисту возможность называть элементы программы и ссылаться на эти элементы с помощью их названия. В языке Ассемблера используются три типа символических имен: простое символическое имя, параметр, символическое имя перехода. Кроме того, могут использоваться символические имена, которые представляют собой параметры, соединенные с некоторыми знаками или другими параметрами.

В главах 2—3 под символическим именем (именем), если не оговорено особо, будет пониматься простое символическое имя. Параметр и символическое имя перехода относятся к макросредствам языка Ассемблера. Правила записи, назначение и функции этих типов символических имен подробно рассматриваются в главе 4. Пока рассмотрим более подробно только простое символическое имя.

Простое символическое имя используется для идентификации различных элементов программы. Например, в приведенной ниже последовательности символических операторов Ассемблера простое символическое имя BEGIN называет машинную команду. В одном из последующих операторов это символическое имя ис-

пользуется для выполнения перехода, т. е. выполняется ссылка к простому символическому имени. Простое символическое имя NEW называется константой, а символическое имя OLD называется областью памяти. В поле операндов команды BEGIN есть ссылка на эти простые символические имена.

Название	Операция	Операнды
BEGIN	MVC	OLD, NEW
...	BC	15, BEGIN
NEW	DC	F'9'
OLD	DS	F

Правила записи простых символических имен следующие:

простое символическое имя может содержать от одной до восьми любых латинских букв и (или) цифр (A—Z, @, \$, #, 0—9); первым знаком простого символического имени должна быть буква (знаки @, \$, # используются в качестве букв и могут появиться в начале символического имени. На некоторых устройствах печати и устройствах подготовки данных знаку \$ соответствует знак C);

в простое символическое имя не должны входить специальные знаки.

Следующие простые символические имена записаны правильно:

ALPHA
Z
A12345
READCARD
@B4
\$A1
#56

Примеры неправильной записи простых символических имен:

1F (первый знак не буква)
FIVEREADC (содержит больше восьми знаков)
RTN*2 (содержит специальный знак)
IN AREA (содержит пробел)

Каждое символическое имя, используемое в программе, должно быть определено.

Символическое имя считается определенным тогда, когда оно записывается в поле названия какого-либо оператора исходного модуля. Под определением символического имени понимается присваивание этому имени следующих величин: значения, характеристики длины и признака переместимости.

Определить признак переместимости — это значит определить, является ли символическое имя переместимым или абсолютным. Абсолютное символическое имя — это простое имя, значение которого постоянно и не изменяется при перемещении программы. Пе-

переместимое символическое имя — это простое имя, значение которого изменяется при перемещении программы.

Значением символических имен, называющих команды, константы, области памяти, программные секции, являются адреса самых левых байт полей памяти, соответствующих этим элементам. Так как при перемещении программы адреса этих элементов могут изменяться, то символические имена, называющие их, являются переместимыми. Значение символического имени должно быть целым положительным числом и не должно превышать $2^{24}-1$.

Характеристика длины имени — это длина в байтах области памяти, которая определяется этим именем, или некоторая заранее установленная величина. Например, символическое имя, называющее команду, которая занимает 4 байта памяти, имеет характеристику длины, равную 4. Считается, что символические имена в операторах CSECT, DSECT, EXTRN, LTORG именуют только один байт, поэтому характеристика длины этих имен принимается равной 1.

Следует отметить особенность определения символического имени с помощью оператора EQU. Символическому имени, указанному в поле названия оператора EQU, присваивается значение выражения, указанного в поле операндов этого оператора. Так как значение выражения в поле операндов может быть переместимым или абсолютным значением, то имя из поля названия оператора EQU в зависимости от значения, которому оно эквивалентно, может быть абсолютным или переместимым. Характеристика длины символического имени, именующего оператор EQU, равна характеристике длины первого термина выражения. Если первым термом в выражении является самоопределенный терм или значение счетчика адреса (знак *), то характеристика длины имени из поля названия принимается равной 1.

Рассмотрим приведенный ниже пример на языке Ассемблера.

Название	Операция	Операнды
	BALR	15,0
	USING	*15
	LA	3,TEN
	LH	2,DATE
	AR	2,3
	STH	2,RES
	SVC	14
DATE	DC	H'25'
TEN	EQU	10
RES	EQU	DATE
	END	

Транслятор Ассемблера условно располагает программу, начиная с нулевого адреса, поэтому символическое имя DATE будет иметь значение X'12' (начальный адрес, равный 0, плюс длина всех команд до оператора DATE). X'12' — это адрес самого левого байта поля, содержащего константу, определяемую оператором

с именем DATE. Для выполнения программа почти всегда загружается с адреса, отличного от установленного транслятором Ассемблера, поэтому фактическое значение символического имени DATE будет другим. Например, если программа будет помещена в память, начиная с адреса X'3000', то значение имени DATE будет равно X'3012'. Таким образом, символическое имя DATE является переместимым символическим именем, так как его значение изменяется при перемещении программы. Символическое имя DATE именуется константой длиной в 2 байта, поэтому характеристика длины имени DATE равна двум.

Символическое имя TEN определяется с помощью оператора EQU. Ему присваивается значение 10. Это десятичный самоопределенный терм, его значение при перемещении программы не изменяется. Следовательно, TEN — абсолютное символическое имя. Характеристика длины TEN равна 1 (характеристика длины самоопределенного термина). Символическое имя RES, также определенное оператором EQU, является переместимым символическим именем (его значение равно значению имени DATE и поэтому изменяется при перемещении программы). Значение и характеристика длины имени RES равны значению и характеристике длины имени DATE.

При написании программы на языке Ассемблера программист должен помнить следующее:

а) каждое символическое имя, используемое в поле операндов символических операторов, должно быть определено. Если в поле операндов оператора будет указано символическое имя (исключая внешнее символическое имя, определяемое в другом модуле), которое не определяется в поле названия в операторах данной программы, то этот оператор является неправильным;

б) каждое символическое имя должно быть определено в данной программе только один раз, т. е. только один раз появиться в поле названия. Исключение составляют имена, называющие операторы START, CSECT, DSECT. Эти имена могут быть использованы более одного раза как имя программной секции или фиктивной области, так как кодирование программной секции или фиктивной области может быть приостановлено и затем продолжено в любой последующей точке. Операторы CSECT, DSECT, которые продолжают секции и фиктивные области, должны быть названы тем же самым именем, которым первоначально именовалась секция и фиктивная область. Такое повторное появление символического имени не считается его повторным определением;

в) символические имена в поле операндов операторов EQU, ORG, CNOP, а также в выражениях, используемых при записи модификаторов в операторах DC, DS, должны быть предварительно определены. Это значит, что символические имена перед использованием их в операндах этих операторов должны появиться в поле названия одного из предшествующих операторов. Например, рассмотрим следующую последовательность команд:

Название	Операция	Операнды
B	LA	5,100
OSH	MVC	DAL,A
SYM1	MVC	DAL,B
SYM2	EQU	SYM1
D	EQU	DAL
DAL	DS	CL50
	...	

Символическое имя A, используемое в поле операндов оператора с именем OSH, не присутствует в поле названия приведенных операторов. Если это имя не появится в поле названия ни в одном из операторов всей программы, оно будет неопределенным, а это значит, что оператор с именем OSH будет неправильным. Последовательность операторов SYM1 и SYM2 правильная, так как имя SYM1, используемое в поле операндов первого оператора EQU, является именем, определенным в предыдущей команде MVC.

Если эти же операторы записать в обратном порядке, то это будет пример неправильной последовательности операторов. Оператор с именем D является неправильным, так как символическое имя DAL предварительно не определено. В поле операндов оператора с именем SYM1 символические имена использованы правильно.

2.1.3. Значение счетчика адреса

Для распределения памяти операторам исходной программы транслятор Ассемблера ведет счетчик адреса. В начале трансляции транслятор устанавливает первоначальное значение счетчика адреса. После обработки каждого оператора программы счетчик адреса увеличивается на длину транслируемого оператора и будет указывать доступный адрес для следующего оператора. Таким образом, во время трансляции каждой машинной команде, константе или области памяти, используемой в транслируемой программе, присваивается определенное значение счетчика адреса. Если необходимо, счетчик адреса может быть установлен транслятором на границу полуслова, слова или двойного слова. Это обозначает, что значение счетчика адреса увеличивается для того чтобы оно было кратно двум, четырем или восьми. Максимальное значение счетчика адреса равно $2^{24}-1$.

Если оператор назван именем, то значением этого имени является значение счетчика адреса, присвоенное этому оператору. Например, предположим, что значение счетчика адреса равно шестнадцатеричному числу 1000, когда транслятор Ассемблера начинает обрабатывать следующий символический оператор:

Название	Операция	Операнды
BEGIN	MVC	A,B

MVC — это мнемонический код операции команды, выполняющей пересылку содержимого области В в область А.

Транслятор присваивает символическому имени BEGIN значение счетчика адреса, равное X'1000'. Команда MVC имеет длину 6 байт (длина команды формата SS), поэтому новое значение счетчика адреса после обработки команды MVC будет равным X'1006'.

Если программа состоит из нескольких программных секций, то транслятор Ассемблера формирует счетчик адреса отдельно для каждой программной секции модуля и работает с каждым счетчиком, как было описано выше.

В языке Ассемблера значение счетчика адреса может быть использовано в качестве термина. Это значит, что программист может воспользоваться текущим значением счетчика адреса в любом месте его программы. Для этого он должен записать в поле операндов оператора знак *. Использовать знак * в операнде машинной команды — это то же самое, что поместить имя в поле названия оператора и затем использовать это имя в операнде оператора. Например, одним из операндов следующей команды является имя этой же команды:

Название	Операция	Операнды
MVI	MVC	P1, MVI

Если в качестве операнда этой команды использовать значение счетчика адреса, то ее можно не именовать:

Название	Операция	Операнды
	MVC	P1, *

Характеристика длины счетчика адреса равна длине оператора, в котором она используется. Характеристика длины счетчика адреса, используемого в операторах EQU, CNOP, ORG и в модификаторах операторов DC и DS, равна 1.

Значение счетчика адреса является переместимым термом, потому что при перемещении программы его значение изменяется. Программист с помощью специальных операторов языка Ассемблера может изменять значение счетчика адреса. Эти возможности рассматриваются в 3.5.

2.1.4. Ссылка на характеристику длины

Под ссылкой на характеристику длины понимается терм, значение которого равно характеристике длины имени или счетчика адреса (допускается ссылка на длину только этих термов). Как уже говорилось, характеристика длины имени \rightarrow это длина в байтах области памяти, которая определяется этим именем, или заранее установленная величина. Характеристика длины значения счетчика адреса (знака *) равна длине команды, в которой этот терм появился. Исключение составляет характеристика значения счетчика адреса в операторах EQU, CNOP, ORG и в модификаторах операторов DC и DS, где характеристика длины значения счетчика адреса принимается равной 1.

Для использования ссылки на характеристику длины как термина необходимо перед именем или перед знаком * написать апостроф, а перед апострофом букву L, например L'NAME. Характеристика длины имени NAME будет значением этого термина.

Ссылка на характеристику длины является абсолютным термом, так как значение, которое этот терм представляет, не изменяется при перемещении программы.

Рассмотрим на примере использование ссылки на характеристику длины как термина. Допустим, необходимо в начале и в конце определенной области памяти разместить заданную константу. Это можно выполнить при помощи следующих операторов:

Название	Операция	Операнды
A1	DS	CL8
B2	DC	CL2'AB'
BBEG	MVC	A1(L'B2),B2
BEND	MVC	A1+L'A1-L'B2(L'B2),B2

Оператор DS определяет область памяти длиной восемь байт. Имя A1 именуется эту область, ему присваивается характеристика длины, равная восьми. Имя B2 именуется знаковую константу длиной два байта, ему присваивается характеристика длины, равная двум.

Оператор с именем BBEG пересылает содержимое поля B2 (константу) в два самых левых байта поля A1 (в начало области). Терм L'B2, заключенный в скобки, представляет поле длины. Значение термина L'B2, равное двум (характеристика длины имени B2), указывает число пересылаемых байт.

Оператор с именем BEND пересылает содержимое поля B2 (константу) в самые правые байты поля A1 (в конец области). Адрес этих байт определяется выражением $A1 + L'A1 - L'B2$, в котором к адресу области A1 прибавляется ее длина, определяемая термом L'A1, и вычитается длина пересылаемой константы, определяемая термом L'B2. Терм L'B2 в скобках представляет собой поле длины, как в операторе с именем BBEG.

Для выполнения действия над операндами, которые находятся в основной памяти, в машинной команде указываются адреса этих операндов, а не сами операнды. Однако если операндом машинной команды является константа, то язык Ассемблера позволяет указывать в команде сам операнд, который участвует при выполнении операции, а не его адрес, как требует формат машинной команды. В этом случае не нужно определять константу в программе с помощью оператора DC. Такая константа, записанная в самой команде вместо ее адреса, называется литералом. При записи константы в машинной команде ей должен предшествовать знак =.

Транслятор Ассемблера при трансляции команды с литералом размещает константу, определенную данным литералом, в некотором месте памяти и помещает адрес поля памяти, содержащего эту константу, в поле операндов обрабатываемой команды. Характеристика длины литерала равна длине константы, определяемой этим литералом.

Литерал является переместимым термом, так как адрес литерала, а не сам литерал будет транслироваться в команду, использующую литерал. Этот адрес изменяется при перемещении программы.

Использование литералов при написании программ на языке Ассемблера подробно рассматривается в 3.7.

2.1.6. Составление и вычисление выражений

Как уже говорилось, операнды операторов языка Ассемблера записываются в виде выражений, которые состояются из рассмотренных ранее термов. В качестве арифметических операций в выражении могут использоваться операции сложения (+), вычитания (—), умножения (*) и деления (/). Для указания последовательности выполнения действий в выражениях могут использоваться скобки, например:

$10 + \text{NAME} * (\text{BETA} - \text{GAMMA})$

При составлении выражений следует соблюдать следующие правила:

а) выражение не должно начинаться со знака арифметической операции (+, —, /, *). Например, выражение $-B + \text{NAME}$ является неправильным. Выражение $0 - B + \text{NAME}$ является правильным;

б) выражение не должно содержать более 16 термов;

в) в выражении не должно быть более пяти уровней скобок (уровень скобок — это левая круглая скобка и соответствующая ей правая круглая скобка);

г) выражение, состоящее из нескольких термов, не должно содержать литералы;

д) нельзя выполнять операцию умножения или деления над переместимыми термами.

Примеры правильно составленных выражений:

```
POLE+X'2D'  
*  
A—25  
*—30  
(FIELD—NTR+2)+GO  
=F'45'  
ALPHA—BETA/(10+AREA*L'FIELD)—10  
B*(B*(B*(B+1)+3*(B—3)))  
APLHA*10  
B'10101111'  
C'ABC'+29  
L'FIELD  
TET/TWO
```

Примеры неправильно составленных выражений:

—A+BAL	начинается со знака арифметической операции
10—/X'02'	два знака арифметических операций подряд
10AB	два термина подряд
=F'20'+10	литерал не единственный в выражении
5+ (—L'NAME+100)	два знака арифметических операций подряд
((A+(B—C)*10)*2+D)	несоответствие скобок

При вычислении выражений выражению, состоящему из одного термина (например, 29, BETA, *, L'SYMBOL, =F'01'), присваивается значение этого термина, а выражение, состоящее из нескольких терминов (например, BETA+10, ENTRY—NAME, 100/10—4*50), приводится к одиночному значению по следующим правилам:

а) каждый терм заменяется своим значением;

б) выполняются слева направо все арифметические операции, указанные в выражении. Умножение и деление выполняются перед сложением и вычитанием, например, $A+B*C$ вычисляется как $A+(B*C)$, но не как $(A+B)*C$. Вычисленный результат является значением выражения;

в) результатом деления всегда является целое число, любая дробная часть результата отбрасывается. Например, $1/2*10$ дает в результате нуль, в то время как $10*1/2$ дает в результате 5;

г) допускается деление на нуль, результатом такого действия будет нуль;

д) выражение в скобках обрабатывается раньше остальных терминов. Например, в выражении $A+B*(C—10)$ выражение $C—10$ вычисляется первым, и полученный результат используется в вычислении окончательного значения;

е) если выражение содержит несколько уровней скобок, то в первую очередь вычисляется самая внутренняя последовательность терминов в скобках. Например, в выражении $A*(B+C*(D+E)+15)$ сначала будет вычислено $(D+E)$, а затем $(B+C*(D+E)+15)$;

ж) окончательные значения выражений должны находиться между -2^{24} и $2^{24}-1$ включительно. Промежуточные результаты

вычислений могут иметь значения между -2^{31} и $2^{31}-1$ включительно. Как исключение, в адресных константах типа А окончательное значение выражения может находиться между -2^{31} и $2^{31}-1$:

з) отрицательные результаты, получающиеся при вычислении выражений, представляются в двончном дополнительном коде.

Каждому выражению условно присваивается характеристика длины. За характеристику длины выражения принимается характеристика длины первого (или единственного) термина в выражении. Если этим термом является самоопределенный терм или ссылка на характеристику длины, то характеристика длины этих термов считается равной 1. Характеристика длины литерала принимается равной длине константы, определяемой литералом. О характеристике длины символического имени и значения счетчика адреса см. в 2.1.4.

2.1.7. Типы выражений

В языке Ассемблера применяются абсолютные и переместные выражения. Выражение называется абсолютным, если его значение не изменяется при перемещении программы. В противном случае выражение называется переместным.

Переместимые выражения разделяются на простые переместимые и составные переместимые выражения. В операторах Ассемблера в основном используются простые переместимые выражения. Составные переместимые выражения используются только в адресных константах. Поэтому иногда вместо термина «простое переместимое выражение» будет употребляться термин «переместимое выражение».

Абсолютное выражение. Абсолютное выражение — это абсолютный терм или арифметическая комбинация абсолютных термов. Абсолютный терм может быть абсолютным символическим именем, самоопределенным термом или ссылкой на характеристику длины. Над абсолютными термами допускается выполнение всех арифметических операций, результат при этом будет абсолютным. Например, выражение $100/X'02' - 5*B'010'$ является абсолютным. Оно содержит абсолютные термы, соединенные знаками арифметических операций. Значение его, равное 40, не изменится при любом расположении в памяти программы, где оно будет использовано.

Выражение может быть абсолютным в некоторых случаях даже тогда, когда в нем используются переместимые термы. В этом случае для того чтобы выражение было абсолютным, необходимо выполнение следующих условий:

а) выражение должно содержать четное число переместимых термов:

б) все переместные термы, входящие в выражение, должны быть спарены. Переместные термы называются спаренными, если они, во-первых, имеют одну и ту же переместимость, т. е.

принадлежат к одной и той же программной секции исходного модуля и, во-вторых, имеют противоположные знаки. Например, в выражении $NAME + X'100' - NAME1$ термы $NAME$ и $NAME1$, которые являются переместимыми символическими именами из одной программной секции и имеют противоположные знаки, будут спарены.

Спаренные переместимые термы могут присутствовать в абсолютных выражениях, потому что в этом случае уничтожается эффект переместимости, т. е. значение, представляемое спаренными термами, остается постоянным, несмотря на перемещение программы. Например, в абсолютном выражении $(A - Y + X)$ A является абсолютным термом, а X и Y — переместимыми термами с одной и той же переместимостью. Если значение A равно 50, значение X равно 25 и значение Y равно 10, то значение выражения равно 65. При перемещении программы на величину 100 значения X и Y станут соответственно равными 125 и 110, но значение выражения останется равным 65 ($50 - 100 + 125$). Значение счетчика адреса может быть спарено с любым другим переместимым термом из той программной секции, где используется значение счетчика адреса.

Использование символических имен из других исходных модулей, т. е. внешних имен, в абсолютных выражениях не допускается, так как внешние имена не могут быть спарены.

Примеры абсолютных выражений (A — абсолютный терм, $X, Y, *$ — переместимые термы с одной и той же переместимостью):

$$\begin{aligned} &A - Y + X \\ &A + 100 \\ &A * A - 5 \\ &X - Y + A \\ &* - X \\ &3 * (X - Y + A) - 5 \end{aligned}$$

Простое переместимое выражение. Простое переместимое выражение — это переместимый терм или арифметическая комбинация переместимых и абсолютных термов, если выполнены следующие условия:

- выражение содержит нечетное число переместимых термов;
- все переместимые термы, кроме одного, спарены;
- неспаренному переместимому терму предшествует знак плюс.

В простом переместимом выражении может присутствовать как неспаренный терм одно внешнее символическое имя. Значением простого переместимого выражения является значение неспаренного переместимого терма, скорректированное значениями абсолютных и (или) спаренных переместимых термов, входящих в выражение.

Например, имеется выражение $Y - X + Y + 10$. В этом выражении Y и X — переместимые термы с одной и той же переместимостью. Если первоначальное значение Y равно 10, значение X равно 5, то значение выражения равно 25. При перемещении программы это значение изменяется. Если значение переместимости равно 100

(т. е. программа переместилась на 100 байт), то значение выражения равно 125. Заметим, что значение выражения $Y-X$, состоящего из спаренных термов, остается равным 5, несмотря на перемещение. Таким образом, новое значение выражения, равное 125, есть новое значение неспаренного переместимого термина Y , скорректированное абсолютными значениями $Y-X$ и 10.

При использовании простых переместимых выражений в адресных константах окончательное значение выражения определяется только во время редактирования.

Примеры простых переместимых выражений:

$$\begin{array}{ll} X-23 * A & = F'1000' \\ X-Y+W & A * A+X-Y+W \\ * & X-Y+X \\ W-X+* & Y \end{array}$$

A — абсолютный терм,

$*$, X и Y — переместимые термы с одной и той же переместимостью,

W — переместимый терм с другой переместимостью.

Составное переместимое выражение. Составное переместимое выражение — это выражение, которое наряду с абсолютными и спаренными термами содержит хотя бы один из следующих элементов: неспаренный переместимый терм с отрицательным знаком; несколько положительных и (или) отрицательных неспаренных переместимых термов.

Присутствие абсолютных или спаренных термов в составном переместимом выражении не обязательно.

Составное переместимое выражение допускается только для представления адресных констант типа A и Y . Окончательное значение выражения (значения адресных констант) устанавливается только после редактирования.

Как и в простых переместимых выражениях, каждый неспаренный переместимый терм в составном выражении не может использоваться в операциях умножения и деления. В противоположность простым переместимым выражениям составные переместимые выражения могут содержать несколько внешних символических имен, так как в них допускается присутствие нескольких неспаренных переместимых термов.

С помощью составного переместимого выражения можно определить, например, расстояние между двумя программными секциями после их загрузки в память.

Примеры составных переместимых выражений:

$$\begin{array}{l} A-Y \\ 0-Z \\ X+Y+Z \\ X-Y-Z \\ X+Y \\ X-Z \\ X+V \\ V+W \\ X-Y-Z-V-W \end{array}$$

- X, Y — переместимые термы с одинаковым признаком переместимости,
Z — переместимый терм с другим признаком переместимости,
A — абсолютный терм,
V и W — символические имена из других модулей, т. е. внешние символические имена.

2.2. МАШИННЫЕ КОМАНДЫ

На языке Ассемблера каждая машинная команда имеет свое символическое изображение. Символический формат записи каждой машинной команды подобен машинному формату записи, но не дублирует его. В пределах каждого из форматов записи машинных команд в символическом виде возможны некоторые отклонения от правил записи в машинном формате. Формат записи всех машинных команд ЕС ЭВМ в символическом виде приведен в приложении 2.

Транслятор Ассемблера переводит символическую запись команды на машинный язык. Машинные команды размещаются транслятором Ассемблера на границе полуслов. Если какая-нибудь команда при трансляции потребует выравнивания (помещения на границу), то пропущенные байты заполняются шестнадцатеричными нулями.

Любая машинная команда может быть названа символическим именем. Это имя может быть использовано другими операторами Ассемблера как операнд. Значение символического имени равно адресу левого байта поля, занимаемого данной машинной командой. Характеристика длины имени машинной команды зависит от формата команды: для команд формата RR характеристика длины равна двум, для форматов RX, RS, SI — четырем, для формата SS — шести.

Каждая машинная команда ЕС ЭВМ имеет определенный мнемонический код операции. Мнемонический код операции условно обозначает машинную операцию, которую должна выполнять команда. Для команд переходов предусматриваются расширенные мнемонические коды. Они определяют машинную команду и условия, при которых выполняется переход. Для каждого условия определен свой мнемонический код операции. Например, переход по «больше» имеет мнемонический код операции ВН, переход по «равно» — ВЕ. Все мнемонические коды операций, в том числе и расширенные, приведены в приложении 2.

Операнды машинных команд — это регистры, непосредственные операнды, адреса памяти и т. д. Они представляются с помощью абсолютных или переместимых выражений. Требования, которые накладываются на используемые типы выражений для тех или иных машинных команд, подробно описываются в приложении 2. Выражения, которые представляют адреса памяти в машинных командах, контролируются, находятся ли эти адреса на

границах, необходимых для операндов данной команды. Номера регистров также контролируются, чтобы они определяли регистры, соответствующие машинной команде, например:

а) команды с плавающей точкой должны использовать регистры с номерами 0, 2, 4 или 6;

б) команды двойного сдвига, умножения слов и деления в первом операнде должны определять четный общий регистр.

За операндами в машинных командах могут следовать комментарии, поясняющие данную команду. Ниже приведено несколько примеров машинных команд, записанных на языке Ассемблера. В комментариях указываются форматы команд. Подробно правила записи машинных команд на бланке кодирования рассматриваются в 3.4.1.

Название	Операция	Операнды	
KOM1	LR	1,2	FORMAT RR
MAD	SVC	TEN	
GAMMA1	L	2,ZETA	FORMAT RX
ALP	L	REG1,39(4,TEN)	
ALP1	SLL	REG2,15	FORMAT RS
ALPHA2	BXH	REG1,REG2,ZETA	
NAME	CLI	ZETA,C'A'	FORMAT SI
GAMMA	SIO	4Ø(9)	
NAME1	MVC	FIELD2,FIELD1	FORMAT SS
NAME2	MVC	4Ø(9,8),3Ø(7)	

2.3. КОМАНДЫ АССЕМБЛЕРА

В языке Ассемблера существуют специальные операторы (команды Ассемблера), с помощью которых можно управлять действиями транслятора Ассемблера при трансляции исходного модуля. Транслятор Ассемблера, встречая во время трансляции команду Ассемблера, выполняет действие, которое она определяет. Например, по оператору EJECT транслятор начинает вывод распечатки результатов трансляции с новой страницы.

Некоторые операторы, например DC, DS, вызывают резервирование областей памяти для констант и других данных, они влияют на значение счетчика адреса. Другие операторы, например TITLE и SPACE, действуют только во время трансляции. Для таких операторов в машинной программе не строятся никакие машинные команды, и они не влияют на значение счетчика адреса.

Команды Ассемблера можно разделить на три группы: команды определения, команды секционирования и соединения и команды управления.

Команды определения:

- EQU — присвоить значение;
- DC — определять константу;

DS	— определить память;
CCW	— определить команду ввода—вывода;
USING	— определить регистр базы;
DROP	— отметить регистр базы.

Команды секционирования и соединения:

START	— определить начало программы;
CSECT	— определить программную секцию;
DSECT	— определить фиктивную область;
ENTRY	— определить входное имя;
EXTRN	— определить внешнее имя;
COM	— определять общую область.

Команды управления:

TITLE	— идентифицировать вывод;
EJECT	— начать новую страницу;
SPACE	— пропустить строку;
PRINT	— управлять печатью;
ICTL	— управлять форматом ввода;
ISEQ	— проверить нумерацию карт;
ORG	— установить счетчик адреса;
LTORG	— начать область литералов;
CNOP	— установить границу;
COPY	— копировать книгу;
END	— закончить модуль;
PUNCH	— перфорировать карту;
REPRO	— перфорировать следующую карту.

2.3.1. Команды определения

К командам определения относятся команды определения регистра базы — USING и DROP; команды определения данных — DC, DS, CCW и оператор EQU, определяющий значение, характеристику длины и переместимость символического имени.

Адреса памяти, используемые в машинных командах, транслятор Ассемблера представляет в виде регистра базы и смещения. Функция транслятора Ассемблера — определять регистр базы и смещение — освобождает программиста от необходимости представлять каждый адрес в виде этих элементов. Для использования этой функции транслятора программисту предоставлены команды Ассемблера USING и DROP. Оператор USING указывает, какие общие регистры может использовать транслятор в качестве регистров базы. Он также указывает транслятору значения адресов, которые должны находиться в этих регистрах в момент выполнения программы (базовые адреса). Оператор USING не загружает базовые адреса в указанные регистры. Загрузка регистров базы программным способом должна быть выполнена программистом.

Команды определения данных предназначены для записи констант, резервирования областей памяти, определения содержимого команд ввода—вывода. Эти команды определения могут быть названы символическими именами, чтобы другие операторы программы могли обращаться к полям, определяемым этими операторами.

Для определения констант используется оператор DC. Один оператор DC может определять одну или несколько констант. В языке Ассемблера допускаются различные типы констант: с фиксированной точкой, с плавающей точкой, десятичные, шестнадцатеричные, знаковые и адресные. Для определения областей памяти используется оператор DS. С помощью оператора DS можно зарезервировать область памяти и присвоить этой области символическое имя (например, можно определить область ввода—вывода или рабочую область). Оператор DS только резервирует область памяти, но не влияет на ее содержимое. Оператор CSW представляет символическую форму записи команды ввода—вывода.

2.3.2. Команды секционирования и соединения

Часто бывает удобно, а иногда и необходимо писать большие программы по частям. Части можно транслировать и отлаживать отдельно, а затем, после устранения всех ошибок, объединять в одну выполняемую программу.

Язык Ассемблера предоставляет возможность разбивать программу на части, называемые программными секциями, и устанавливать между ними символические связи. Можно транслировать самостоятельно каждую секцию отдельно или несколько секций вместе. Соединение отдельно транслируемых секций в одну выполняемую программу и установление между ними связей, которые были определены программистом символически, выполняется Редактором. Разбивать программу на секции можно с помощью операторов START и CSECT, которые идентифицируют программную секцию.

Язык Ассемблера предусматривает средства символических связей отдельно транслируемых программ или секций одного исходного модуля. Для этого используются операторы DSECT, COM, EXTRN, ENTRY и константы типа V.

Транслятор Ассемблера готовит специальную информацию исходя из указаний команд секционирования и соединения, присутствующих в данном транслируемом модуле, с помощью которой Редактор соединит в дальнейшем все независимо транслируемые части программы в готовую для выполнения программу.

2.3.3. Команды управления

Команды управления используются для управления трансляцией. Операторы TITLE, EJECT, SPACE, PRINT управляют выводом распечатки результатов трансляции. Оператор TITLE дает возможность идентифицировать распечатку и выходные карты объектного модуля, он вызывает переход к новой странице распечатки и печать заглавия в начале этой и последующих страниц. Оператор EJECT вызывает печать следующего за ним текста с новой страницы, а оператор SPACE — печать пустых строк в рас-

печатке. Оператор PRINT управляет содержанием распечатки программы. С помощью этого оператора можно отменить вывод распечатки, печатать или не печатать операторы, вставляемые в программу по макрокомандам, печатать полностью или частично объектный код констант, построенный транслятором.

Операторы ICTL и ISEQ управляют вводом исходного модуля. Оператор ICTL определяет формат вводимых исходных карт, оператор ISEQ проверяет последовательность входных карт. С помощью операторов PUNCH и REPRO можно отперфорировать в объектный модуль карты с определенной информацией. Операторы ORG и CNOP воздействуют на счетчик адреса: с помощью оператора ORG можно установить определенное значение счетчика адреса, оператор CNOP позволяет установить значение счетчика адреса на нужную границу. Оператор COPY указывает транслятору на необходимость вставить во время выполнения трансляции ранее написанную часть исходной программы. Оператор LTORG определяет в программе область памяти для размещения литералов. Оператор END указывает конец исходного модуля, он должен быть самым последним оператором в исходном модуле.

Ни один из операторов управления трансляцией, за исключением COPY и CNOP, не порождает в транслируемой программе команд или констант.

2.4. МАКРОСРЕДСТВА

Кроме упомянутых ранее элементов языка, язык Ассемблера предоставляет программисту некоторые более широкие возможности, чем кодирование на уровне машинного языка. Эти возможности составляют макросредства Ассемблера, которые представляют собой набор определенных операторов языка Ассемблера. Макросредства позволяют:

- вставлять в исходный модуль набор операторов;
- изменять последовательность вставляемых операторов;
- изменять отдельные части операторов.

Макросредства упрощают кодирование программ, сокращают количество ошибок при программировании и обеспечивают использование стандартных последовательностей операторов для выполнения некоторых функций.

Макросредства языка Ассемблера подробно рассматриваются в главе 4.

2.5. КОММЕНТАРИИ

Операторы комментариев предназначены для записи пояснений к исходной программе, которые могут быть полезны во время отладки и эксплуатации программы. Операторы комментариев

не оказывают влияния на транслируемую программу, они могут записываться в любом месте исходной программы. Можно записывать любое количество поясняющего текста, используя для этого группу операторов комментариев.

Имеются два типа операторов комментариев. Операторы комментариев первого типа начинаются со знака *. Этот тип операторов комментариев используется для записи комментариев в любом месте исходной программы. Операторы комментариев другого типа начинаются с точки, за которой следует знак *. Эти операторы используются для записи комментариев только в макроопределениях.

РЕКОМЕНДАЦИИ ПО ПРОГРАММИРОВАНИЮ

3.1. ЗАПИСЬ ОПЕРАТОРОВ ЯЗЫКА АССЕМБЛЕРА

3.1.1. Алфавит языка Ассемблера

В алфавит языка Ассемблера входят следующие знаки кода ДКОИ:

латинские буквы от A до Z;

знаки \$, #, @, которые используются как буквы;

цифры от 0 до 9;

специальные знаки: +, —, =, *, (,), ', /, &, ., пробел и запятая.

Кроме того, все знаки кода ДКОИ могут быть записаны в комментариях, в операндах макрокоманд, в строке за оператором REPRO, а также в следующих случаях между парными апострофами: в знаковом самоопределенном терме; в знаковой константе; в знаковом выражении макросредств; в поле операндов операторов TITLE, PUNCH, MNOTE.

3.1.2. Бланк кодирования

Операторы языка Ассемблера записываются на бланке кодирования (см. рис. 5). Правила записи операторов приведены в 3.1.3.

Бланк состоит из строк, разделенных на 80 позиций. Каждая строка перфорируется на отдельной перфокарте. Каждая позиция строки бланка кодирования соответствует колонке на перфокарте.

В верхней части бланка кодирования отведено место, где программист может записать общую информацию о своей программе (например, название программы, дату написания) и указания оператору, перфорирующему карты.

Каждая строка бланка кодирования состоит из двух частей: колонки 1—71 отводятся для записи оператора, колонки 73—80 — для идентификации строки оператора.

Оператор записывается в колонках 1—71 первой строки и, если необходимо, в колонках 16—71 последующих строк продолжения. Колонки 1, 71 и 16 называются соответственно колонкой начала, колонкой конца и колонкой продолжения и представляют стандартные границы для записи оператора. Эти границы могут быть изменены с помощью оператора ICTL (см. 3.1.5).

Если требуется продолжить запись оператора на следующей

строке, необходимо записать любой знак, отличный от пробела, в колонке, следующей за колонкой конца оператора. Эта колонка называется колонкой указателя продолжения. В случае стандартных границ оператора колонкой указателя продолжения является колонка 72. В строке продолжения в колонках слева от колонки продолжения должны быть пробелы. При стандартных границах оператора пробелы должны быть в колонках 1—15 строки продолжения. Допускается одна строка продолжения для каждого оператора в Ассемблере Е и две строки продолжения в Ассемблере F. Исключение составляют макрокоманды и операторы прототипа, которые могут иметь любое количество строк продолжения.

3.1.3. Правила записи операторов

Оператор, записанный на бланке кодирования, может содержать от 1 до 4 полей: поле названия, поле операции, поле операндов и поле комментариев. Поле комментариев и поле названия могут быть пустыми, в поле операции всегда, а в поле операндов, как правило, должна присутствовать нужная запись. Для каждого поля на бланке кодирования отводится соответствующая графа. Располагая поля оператора в графах, обозначенных на бланке кодирования, программист записывает операторы в стандартном формате. Поля операторов в строке бланка кодирования можно записывать и произвольно, в свободном формате, но при этом должны быть выполнены правила записи операторов, изложенные в настоящем пункте.

В поле названия записывается символическое имя оператора, которое выбирается программистом для идентификации оператора. Символическое имя не должно содержать более 8 знаков. При записи оператора в стандартном формате для поля названия отведены колонки 1—8. Записывать символическое имя в поле названия не обязательно, но если оно присутствует, то первый знак имени должен находиться в колонке начала. Пробел в колонке начала обозначает, что оператор не имеет имени. Если оператор располагается в стандартных границах, то поле названия начинается с колонки 1. Программист может использовать в качестве символических имен соединения параметров с другими знаками, тогда поле названия может содержать больше восьми знаков. В этом случае для поля названия отводится столько колонок, сколько их необходимо для размещения всех знаков имени. При составлении названия необходимо помнить, что после генерации сформированное символическое имя не должно содержать больше восьми знаков. Символические имена в поле названия не должны содержать пробелы. Это относится как к именам, которые непосредственно записаны программистом, так и к именам, которые появляются в результате генерации.

Поле операции следует за полем названия и отделяется от него по крайней мере одним пробелом. В поле операции записы-

ается мнемонический код операции, состоящий из пяти или менее знаков для машинных команд, команд Ассемблера и команд генерации и из восьми или менее знаков для макрокоманд. Запись в поле операции является обязательной, и она должна находиться в первой строке оператора. Если у оператора отсутствует имя, поле операции должно начинаться по крайней мере на одну позицию правее колонки начала. При записи оператора в стандартном формате для поля операции отведены колонки 10—14, колонка 9 отделяет поле названия от поля операции и должна содержать пробел. Мнемонический код операции в поле операции не должен содержать пробелы. Это относится и к тем кодам операций, которые появляются в результате генерации.

Поле операндов следует за полем операции и отделяется от него хотя бы одним пробелом. В зависимости от оператора в поле операндов может быть записан один или несколько операндов. Операнды при записи должны разделяться запятыми. Между операндами и запятыми, которые их разделяют, не должно быть пробелов. Сами операнды тоже не должны содержать пробелов, за исключением тех случаев, когда пробел присутствует как знак в самоопределенном терме, в знаковой константе или литерале, определяющем знаковую константу. При записи операторов в стандартном формате для поля операндов отводятся колонки 16—71. Колонка 15 отделяет поле операции от поля операндов и должна содержать пробел. Запись поля операндов можно продолжить в последующих строках — в строках продолжения, в колонках 16—71.

Поле комментариев располагается за полем операндов. Запись комментариев не обязательна. Комментарии — это описательная информация о программе. Для записи комментариев могут быть использованы все знаки кода ДКОИ. Поле комментариев от последнего операнда должно отделяться пробелом и не должно выходить за колонку конца. При записи операторов в стандартном формате для поля комментариев можно использовать те колонки из колонок 16—71, которые остаются свободными после записи поля операндов. Для некоторых операторов запись операндов не обязательна (например, ORG, END). Если для таких операторов требуется записать комментарии, то отсутствие операндов должно быть указано запятой. Запятая от поля комментариев должна быть отделена хотя бы одним пробелом.

Таким образом, при записи операторов на бланке следует соблюдать следующие правила:

все поля оператора всегда должны отделяться друг от друга по крайней мере одним пробелом и должны быть расположены в следующем порядке: поле названия, поле операции, поле операндов, поле комментариев;

каждый оператор должен иметь запись в поле операции и поле операндов (для отдельных операторов запись в поле операндов не обязательна); поле названия и поле комментариев могут быть пустыми;

поле названия, поле операции и хотя бы один пробел, следующий за полем операции, должны находиться в первой строке оператора;

записи в поле названия, поле операции и поле операндов не должны содержать пробелов;

поле названия должно начинаться в колонке начала;

если в колонке указателя продолжения находится пробел, то на следующей строке должен начинаться новый оператор (у данного оператора нет продолжения). Если в колонке указателя продолжения записан любой знак, отличный от пробела, то следующая строка считается продолжением данного оператора;

все поля операторов должны содержаться внутри границ, указанных колонками начала, конца и продолжения;

операторы комментариев записываются без учета деления строки бланка на поля, признак оператора комментариев записывается, начиная с колонки начала.

3.1.4. Примеры записи операторов на бланке кодирования

Рассмотрим возможности записи на бланках кодирования операторов языка Ассемблера на примерах (см. рис. 5). При записи операторов используются стандартные границы, т. е. колонками начала, конца и продолжения являются соответственно колонки 1, 71, 16.

Первые восемь операторов записаны в стандартном формате, т. е. все поля операторов расположены в соответствующих графах на бланке кодирования: поле названия — в колонках 1—8, поле операции — в колонках 10—14, поля операндов и комментариев — в колонках 16—71. В колонках 9 и 15, предназначенных на бланке кодирования для разделения поля названия и поля операции, поля операции и поля операндов, находятся пробелы. Поле комментариев от поля операндов отделяется одним или несколькими пробелами. Имена BEGIN и PROD, записанные в поле названия, начинаются в колонке начала 1. Оператор с именем PROD имеет строку продолжения. Для того чтобы продолжить запись оператора с именем PROD на следующей строке, в колонке указателя продолжения 72 первой строки этого оператора записан знак X. Запись оператора в строке продолжения начинается с колонки продолжения 16. Операторы, которые начинаются со знака *, являются операторами комментариев. Знак * записан в колонке начала.

Девятый и десятый операторы записаны в свободном формате. В девятом операторе запись в поле названия отсутствует, на что указывает пробел в колонке начала 1, с колонки 2 располагается поле операции, а через пробел от поля операции — поле операндов. Поле комментариев этого оператора начинается с колонки 10 и отделено от поля операндов пробелом. Поле операции

десятого оператора (MVI) расположено в колонках 17—19, в поле операндов этого оператора содержится пробел, но, тем не менее, запись правильная, потому что пробел записан в знаковом самоопределенном терме. Хотя рассмотренные два оператора записаны без учета граф, указанных на бланке, они записаны правильно.

Оператор с именем NAME также записан в свободном формате. Имя, записанное в поле названия, начинается в колонке начала 1. Оператор имеет строку продолжения, на что указывает знак X в колонке 72. Поле операции записано в первой строке оператора, хотя и не в графе «Операция» бланка. Поле операндов начинается в первой строке и продолжено на следующей с колонки продолжения 16.

Операторы с именами NAME1 и NAME2 записаны в свободном формате, но в их записи есть ошибки. Имя NAME1 начинается не с колонки начала. В колонке начала записан пробел, поэтому это имя рассматривается транслятором как код операции, который будет недействительным. Оператор с именем NAME2 имеет строку продолжения, но поле операции оператора находится не в первой строке оператора.

Оператор с именем NAME3 и оператор, следующий за ним, записаны в стандартном формате. Но эти операторы являются неправильными, так как в поле названия и в поле операндов присутствуют пробелы, что не допускается правилами записи операторов.

Оператор с именем NAME4 записан в свободном формате. Он имеет строку продолжения, которая записана неправильно: запись в строке продолжения начинается не с колонки продолжения 16, а раньше.

Оператор с именем DEC записан правильно, в свободном формате. В поле операндов этого оператора записан пробел, но в этом случае запись поля операндов правильная, потому что пробел записан в знаковой константе.

В последнем операторе END поле операндов отсутствует (операнд — пробел). Чтобы записать комментарий в этом случае, в поле операндов записана запятая, которая отделена от поля операции и поля комментариев пробелами.

3.1.5. Изменение границ операторов

Стандартные границы оператора, определенные номерами колонок начала, конца и продолжения (1, 71, 16), можно изменить с помощью оператора ICTL.

Оператор ICTL (УПРАВЛЯТЬ ФОРМАТОМ ВВОДА) сообщает транслятору Ассемблера, в каких границах расположены операторы на бланке кодирования и соответственно на перфокартах. Оператор ICTL может использоваться в программе только один раз и должен предшествовать всем другим операторам в исходном модуле. При использовании оператора ICTL все другие

операторы в программе должны записываться в границах, определенных этим оператором. Оператор ICTL имеет следующий формат:

Название	Операция	Операнды
Пробел	ICTL	От одного до трех десятичных чисел в виде: a,b,c

Операнд а указывает колонку начала. Он должен обязательно присутствовать в поле операндов оператора ICTL. Операнд а может принимать значения от 1 до 40 включительно.

Операнд b определяет колонку конца и может принимать значения от 41 до 80 включительно. Операнд b может отсутствовать, в этом случае он считается равным 71. Колонка b+1 является колонкой указателя продолжения и определяет, является ли следующая строка строкой продолжения оператора.

Операнд с указывает колонку продолжения оператора. Он может принимать значения от 2 до 40 включительно. Значение операнда с должно быть больше значения операнда а. В случае отсутствия операнда с строки продолжения не допускаются. Запись каждого оператора в этом случае должна помещаться в одной строке. Если операнд b равен 80, т. е. колонка 80 является колонкой конца оператора, то операнд с не должен присутствовать в операторе ICTL.

В случае неправильной записи оператора ICTL транслятор Ассемблера прекращает трансляцию исходного модуля.

При отсутствии в исходном модуле оператора ICTL границы оператора предполагаются стандартными, как если бы в модуле присутствовал следующий оператор ICTL:

Название	Операция	Операнды
	ICTL	1,71,16

Рассмотрим следующий оператор:

Название	Операция	Операнды
	ICTL	20

Оператор ICTL определяет колонку 20 как колонку начала. Так как колонка конца не указана, то она предполагается равной 71. Карты продолжения не допускаются, так как не определена колонка продолжения.

На рис. 6 приведены примеры записи операторов на бланке кодирования в случае присутствия в программе оператора ICTL. Оператор ICTL в приведенном примере определяет колонку 6 как колонку начала, колонку 55 как колонку конца, колонку 20 как колонку продолжения. Операторы записаны в свободном формате с учетом границ оператора, определенных оператором ICTL. Символические имена BEGIN, NAME, OLD, LAT записаны, начиная с колонки 6, которая указана как колонка начала. Поле операции в операторах расположено произвольно. Там, где нет поля названия, полю операции предшествует один или несколько пробелов, начиная с колонки 6.

При обработке оператора с именем RESULTUT знаками символического имени транслятор будет считать только те знаки, которые начинаются с колонки 6, т. е. символическим именем этого оператора будет имя TUT. Знаки в колонках 1—5 восприниматься не будут. Поэтому используемое в одном из операторов имя RESULTUT будет не определено.

Оператор с именем LAT имеет строку продолжения. Колонкой указателя продолжения является колонка 56, следующая за колонкой конца. В ней записан знак X, который указывает, что дальше следует строка продолжения.

Оператор ICTL определяет колонку 20 колонкой продолжения. В операторе с именем LAT строка продолжения записана правильно. Оператор с именем NAME тоже в колонке указателя продолжения 56 имеет знак, указывающий строку продолжения. Но строка продолжения этого оператора записана неправильно: запись начинается с колонки 16. Нарушено правило, которое требует, чтобы до колонки продолжения, в данном примере до колонки 20, находились пробелы.

3.1.6. Идентификация

Каждая строка, записанная на бланке кодирования, может быть идентифицирована.

Идентификация строки бланка кодирования не является частью оператора и используется для идентификации перфокарт исходной программы и (или) их нумерации. Для записи идентификации на бланке кодирования могут использоваться колонки, следующие за колонкой указателя продолжения до последней колонки строки или от первой колонки строки до колонки начала оператора. Таким образом, если границы оператора стандартные, то для идентификации можно использовать колонки 73—80.

Записывать идентификацию не обязательно. При записи идентификации можно использовать все знаки кода ДКОИ. В операторах, приведенных на рис. 5, идентификация записана в колонках 73—80, как позволяют стандартные границы оператора. На рис. 6, где приведены операторы, записанные в границах, определенных оператором ICTL, в строках бланка тоже записана идентификация. Так как здесь колонкой указателя продолжения явля-

ется колонка 56 (установлена оператором ICTL), то для идентификации используются колонки 57—80.

Язык Ассемблера предоставляет программисту возможность следить за порядком операторов во время ввода исходного модуля. Для этого программист должен в поле идентификации закодировать возрастающую последовательность номеров. Затем нужно с помощью оператора ISEQ, который предназначен для проверки последовательности входных карт, указать транслятору, что необходимо проверить эту последовательность. Оператор ISEQ имеет следующий формат:

Название	Операция	Операнды
Пробел	ISEQ	Пробел или два десятичных числа в виде: a,b

Операнды а и b указывают начальную и конечную колонки из тех колонок на перфокарте, которые должны проверяться при вводе. Операнд b должен быть больше или равен операнду а. Проверяемые колонки не должны находиться между колонками начала и конца оператора.

Обычно идентификация операторов записывается за колонкой указателя продолжения. Если она будет отперфорирована на картах вместе со всей информацией каждой строки бланка кодирования, то при вводе карт идентификация будет проверяться на возрастание, если в программе присутствует оператор ISEQ.

Проверка последовательности на возрастание начинается с карты, следующей за оператором ISEQ. Проверяются битовые значения поля, указанного операндами оператора ISEQ. Ошибки, обнаруженные при проверке, не влияют на результат трансляции, они выдаются в распечатке результатов трансляции. Проверка нумерации карт ведется только для операторов исходного модуля. Проверка нумерации карт, включаемых из библиотеки исходных модулей (при помощи оператора COPY или макрокоманд), не выполняется. Оператор ISEQ с пустым полем операндов прекращает процесс проверки. Проверка может быть возобновлена другим оператором ISEQ.

Операторы, приведенные на рис. 5, имеют идентификацию. В идентификации записан идентификатор программы (PRIM) и возрастающая последовательность номеров. При трансляции нумерация карт будет проверяться, начиная с оператора с номером 17, потому что перед ним записан оператор ISEQ. По этому оператору будут проверяться колонки 77—80. Оператор с номером 23 находится перед оператором с номером 22, поэтому транслятор отметит, что номер оператора 22 не возрастает.

В примерах, приведенных на рис. 6, колонкой указателя продолжения является колонка 56, что установлено оператором ICTL. Колонки 57—80 в этом случае можно использовать для иденти-

фикации. Идентификация записана в колонках 59—62 и 70—75. Оператор ISEQ указывает, что необходимо проверять только колонки 70—75. Оператор 11 будет отмечен транслятором, как нарушающий последовательность, так как в предыдущем операторе первыми двумя буквами идентификатора являются буквы АВ, а не АА (код буквы В больше кода буквы А). Начиная с оператора 13, проверка на возрастание выполняться не будет, так как предыдущий оператор 12 является оператором ISEQ без оператора, что и вызывает прекращение проверки.

3.2. АДРЕСАЦИЯ ПАМЯТИ

Все машинные команды, за исключением команд формата RR, обращаются к основной памяти. Как уже говорилось, единицей основной памяти ЕС ЭВМ является байт. Все байты основной памяти пронумерованы, начиная с 0. Адрес каждого байта численно равен его номеру. Максимальный адрес равен 16 777 215, так что для его указания необходимо 24-разрядное двоичное число. Команды машины оперируют с областями основной памяти, состоящими из одного или более байт. Адресом области основной памяти является адрес самого левого байта этой области.

3.2.1. Представление адреса в ЕС ЭВМ

Адреса памяти в машинных командах ЕС ЭВМ представляют в виде регистра базы и смещения. В командах формата RX дополнительно указывается еще регистр индекса. Действительный адрес памяти формируется сложением базового адреса, т. е. содержимого регистра базы, со смещением. Для команд формата RX к этой сумме добавляется еще содержимое регистра индекса.

Базовый адрес представляет собой содержащееся в общем регистре 24-разрядное двоичное число. Регистр базы задается в команде полем В1 или В2 (см. рис. 4). Базовый адрес участвует в формировании всех адресов основной памяти.

Индекс также представляет собой содержащееся в общем регистре 24-разрядное двоичное число. Регистр индекса задается в команде полем X1 (см. рис. 4). Он входит в состав адреса только для команд формата RX.

Смещение, задаваемое полем D1 или D2 (см. рис. 4), представляет собой 12-разрядное двоичное число, которое располагается в самой команде и может принимать значение от 0 до 4095. Смещение участвует в формировании каждого адреса основной памяти. Оно позволяет адресовать байты памяти, следующие последовательно за байтом, адрес которого равен базовому адресу, или за байтом, адрес которого равен сумме индекса и базового адреса.

При формировании действительного адреса все компоненты складываются как двоичные числа, 24 разряда полученной суммы

являются адресом памяти, при этом ни команда, ни содержимое регистра базы или регистра индекса не изменяются.

При адресации нужно учитывать особенности использования общего регистра 0. Если регистр 0 указывается как регистр базы или регистр индекса, то при формировании действительного адреса используется не его содержимое, а значение 0.

3.2.2. Преимущества адресации с регистром базы

Одним из преимуществ представления адреса памяти в виде регистра базы и смещения является экономия памяти за счет сокращения длины команд. Для указания адреса памяти необходимо 24 двоичных разряда. Но для указания адреса памяти с помощью регистра базы и смещения достаточно 16 двоичных разрядов. Действительно, для указания регистра необходимо 4 разряда. Четырехразрядное поле для регистра базы в команде позволяет указывать один из общих регистров с номерами от 0 до 15. Младшие 24 разряда общего регистра используются для значения базового адреса. Базовым адресом может быть адрес любого байта основной памяти. Для смещения в команде отводится 12 разрядов. Такое смещение позволяет адресовать 4096 байт, начиная с базового адреса. Таким образом, с помощью поля в 16 разрядов можно указывать любой 24-разрядный адрес.

Но основным преимуществом представления адреса памяти в виде базы и смещения является независимость машинной программы от ее положения в памяти. Поскольку адреса памяти в командах задаются регистром базы, а содержимое регистра базы устанавливается во время выполнения программы, то программу (команды и данные) можно размещать в любом месте основной памяти без дополнительной обработки.

Действительно, допустим, что в машинной команде используется область памяти. Адрес области в команде представлен в виде регистра базы и смещения. Для определенности будем считать, что смещение равно S , регистром базы является регистр 4, а базовым адресом является адрес, с которого программа располагается в памяти. Загрузка базового адреса в регистр базы 4 выполняется всегда, когда программа получает управление. Тогда, если программа будет располагаться с адреса A , то адрес используемой области памяти будет равен $A + S$. При выполнении машинной команды адрес этой области вычисляется как сумма содержимого регистра базы 4 и смещения и равен $A + S$. Если же программу расположить с адреса B , то адрес используемой области будет равен $B + S$. В этом случае при выполнении программы в регистр базы 4 будет загружен адрес B , и машинная команда обратится к адресу $B + S$.

Таким образом, адрес, записанный в команде однажды в виде регистра базы и смещения, остается неизменяемым, где бы ни располагалась программа в памяти. При таком способе адресации программа может выполняться в любом месте памяти, для

чего достаточно загрузить в регистр базы правильный базовый адрес. Этот факт особенно важен в современных операционных системах, которые могут загружать программу для выполнения в любое место памяти, которое программисту может быть даже неизвестно.

3.2.3. Способы адресации в Ассемблере

На языке Ассемблера адреса памяти можно записывать так, как этого требует формат машинной команды — в виде регистра базы и смещения. Однако для программиста такой способ записи адресов очень неудобен. Обычно при программировании на языке Ассемблера используются символические имена действительных адресов памяти, т. е. номеров байт памяти. Представление действительного адреса в виде регистра базы и смещения, как требует формат машинной команды, осуществляется транслятором. Транслятор выбирает необходимый регистр базы и вычисляет смещение. Для этого программист должен указать транслятору, каким регистром должна базироваться каждая область памяти и какие базовые адреса должны находиться в регистрах базы. Программист, составляя программу, должен позаботиться о том, чтобы регистр базы содержал правильный базовый адрес во время выполнения программы.

Адреса явные и неявные. Рассмотрим подробнее способы представления адреса на языке Ассемблера. Адреса памяти используются в основном как операнды машинных команд. В зависимости от способа их представления адреса разделяются на явные и неявные.

Адрес, представленный в команде в виде регистра базы и смещения, называется явным. Для записи смещения и номера регистра базы должны использоваться абсолютные выражения.

Наряду с явными адресами программист может использовать действительные адреса памяти. Эти адреса памяти задаются переместимыми или абсолютными выражениями. Адрес, представленный программистом таким образом, называется неявным. Абсолютное выражение определяет абсолютный неявный адрес, переместимое выражение — переместимый неявный адрес. Для неявного адреса транслятор Ассемблера определяет регистр базы и вычисляет смещение и тем самым формирует явный адрес из неявного.

В приложении 2 приведены оба способа записи адресов для всех машинных команд форматов RX, RS, SI и SS.

В приведенных символических операторах в поле комментариев указан формат каждой команды. В команде IC (ПРОЧИТАТЬ СИМВОЛ) записан явный адрес байта, который нужно прочитать. В явном адресе абсолютными выражениями указаны: регистр базы — X'F', регистр индекса — 2, смещение — 400. В командах LA (ЗАГРУЗКА АДРЕСА) записан неявный адрес. В первой команде LA используется регистр индекса. Символическому имени REGI, указывающему регистр индекса, должно быть присвоено

абсолютное значение в пределах от 0 до 15. В команде SLA (СДВИГ ВЛЕВО) записан явный адрес: регистр базы — 4, смещение — 500. В команде SRA (СДВИГ ВПРАВО) записан неявный адрес. Символическое имя AOBL, определяющее адрес в этой команде, может быть абсолютным или переместимым. В командах MVI (ПЕРЕСЫЛКА), которые пересылают непосредственный операнд B'00001111' по указанному адресу памяти, в одном случае записан явный адрес, в другом — неявный. Явный адрес записан в виде регистра базы с номером X'A'(10) и смещения, равного 50. Неявный адрес задан символическим именем BAIT. В первой команде MVC (ПЕРЕСЫЛКА) первый адрес записан явный, второй — неявный. В явном адресе регистр базы указан символическим именем REGB. Во второй команде MVC записаны два неявных адреса.

Название	Операция	Операнды
	IC	1,400(2,X'F') RX
	LA	2,SIMB(REGI) RX
	LA	2,ADR RX
	SLA	1,500(4) RS
	SRA	1,AOBL RS
	MVI	BAIT,B'00001111' SI
	MVI	50(X'A'), B'00001111' SI
	MVC	20(5,REGB),ADR SS
	MVC	ADR1,ADR SS

Длины явные и неявные. При представлении команд формата SS на машинном языке указывается общая длина обоих операндов или длина каждого операнда в байте 2 первого полуслова, занимаемого командой. Эта длина (длины) определяет длину областей памяти, над содержимым которых необходимо выполнить операцию. При программировании на языке Ассемблера эту длину может указывать программист, тогда транслятор использует указанную длину при построении машинной команды. Если программист не указал длину, то ее вычисляет транслятор Ассемблера.

Если в записи адреса (явного или неявного) присутствует длина, то это значит, что указывается явная длина операнда. Отсутствие длины в записи адреса обозначает неявное задание длины, т. е. неявную длину. В этом случае транслятором Ассемблера длина принимается равной одной из следующих величин:

характеристике длины выражения, определяющего смещение, если задан явный адрес;

характеристике длины выражения, определяющего адрес, если задан неявный адрес.

Как уже говорилось, характеристикой длины выражения является характеристика длины самого первого термина этого выра-

жения. Явная длина записывается программистом в операнде в виде абсолютного выражения.

В машинных командах на языке Ассемблера всегда указывается действительная длина операнда. Например, если команда должна обработать 6 байт, то нужно указать в качестве длины число 6. Но значение, которое помещает транслятор в поле длины построенной команды на машинном языке, будет на единицу меньше, потому что этого требует формат машинной команды. Если машинная команда должна обрабатывать операнд, состоящий из одного байта, то длина в символической команде может указываться равной нулю или единице.

Рассмотрим использование явной и неявной длины на примерах:

Название	Операция	Операнды
ADR	MVC	200(X'10',2),400(2)
ADRI	MVC	X'200'(2),FIELD
	CLC	ADR4,ADR3
	AP	200(4,2),400(B'010',2)
	CP	ADR4,ADR3
	CP	ADR2(2),ADR3
FIELD	DC	16X'00001'
ADR3	DC	4X'00000000'
ADR4	DC	P'1234567'

В примерах записаны символические команды формата SS. В команде MVC с именем ADR записаны явные адреса и указана явная длина (в команде MVC используется общая длина для двух операндов). Длина указывается с помощью шестнадцатеричного самоопределенного термина X'10'.

В команде MVC с именем ADRI первый адрес записан явный, а второй — неявный. Длина в этой команде не указана, используется неявная длина первого операнда. Первый операнд задается явным адресом, поэтому длина принимается равной характеристике длины выражения, определяющего смещение. Смещение записано в виде шестнадцатеричного самоопределенного термина X'200'. Характеристика длины самоопределенного термина равна 1. Значит, неявная длина равна 1.

В команде CLC (СПРАВНЕНИЕ КОДОВ) записаны два неявных адреса. В этой команде тоже используется общая длина для двух операндов. В данном случае будет использоваться неявная длина первого операнда, которая равна характеристике длины имени ADR4, т. е. 4.

В команде AP (СЛОЖЕНИЕ ДЕСЯТИЧНОЕ), где требуется длина двух операндов, записаны два явных адреса, указана явная длина. Для первого операнда указана длина, равная 4. Для второго операнда длина, равная 2, указана в виде двоичного термина.

В двух командах СР (СРАВНЕНИЕ ДЕСЯТИЧНОЕ) используются неявные адреса. Команда СР требует указания длины для каждого операнда. В первой команде СР используется неявная длина каждого операнда. Неявная длина первого операнда равна характеристике длины имени АDR4, второго—характеристике длины имени АDR3. Для обоих операндов неявная длина равна 4. Во второй команде для первого операнда указана явная длина, равная 2. Для второго операнда используется неявная длина, равная 4.

Относительная адресация. Язык Ассемблера предоставляет программисту возможность пользоваться относительной адресацией. Относительная адресация представляет собой способ адресации команд и областей данных путем определения их адреса относительно значения счетчика адреса или другой команды или области данных, которые имеют символическое имя. Относительная величина всегда указывается в байтах.

В следующем примере¹ в некоторых командах используется относительная адресация вместо того, чтобы определять новые символические имена:

Название	Операция	Операнды	Идентификация
NAME	BALR	15,Ø	1
	USING	*,15	2
	L	2,DATE	3
	LA	3,*+8	4
	S	2,DATE+4	5
	LA	4,5	6
	CR	2,4	7
	BL	PERE+2	8
	S	2,DATE+8	9
	BR	3	10
PERE	ST	2,DATE+12	11
DATE	SVC	14	12
	DC	F'26'	13
	DC	F'15'	14
	DC	F'5'	15
	DS	F'Ø'	16
	END		17

Оператор 10 является командой перехода по регистру. По этой команде выполняется переход по адресу, находящемуся в регистре 3 (на оператор 6). Оператор 4 загружает в регистр 3 необходимый адрес. Это адрес записан в виде *+8 и на 8 байт больше, чем текущее значение счетчика. Это и есть адрес оператора 6. Конечно, можно было поместить в поле названия оператора 6 символическое имя и это имя использовать в операторе 4, но в данном случае используется относительная адресация.

¹ Здесь и далее в объяснениях к примерам используются номера операторов, записанных в поле идентификации.

В команде 5 адрес данного указывается относительно адреса DATE. По адресу DATE находится константа, имеющая длину 4 байта. В команде 5 используется адрес DATE+4, поэтому происходит обращение к константе, определенной оператором 14. Аналогично записано обращение к константам, определяемым операторами 15 и 16 в операторах 9 и 11.

В команде 8 указывается адрес команды 11. Этот адрес указывается в виде выражения PERE+2 (длина команды 10 равна 2 байтам). Если бы потребовалось обращение к команде 7, то его можно было бы записать, например, такими двумя способами, не называя эту команду именем: NAME+8 и PERE-10.

3.3. ИСПОЛЬЗОВАНИЕ ОПЕРАТОРОВ USING И DROP

Как отмечалось ранее, команды ЕС ЭВМ, записанные на машинном языке, не содержат действительных адресов данных. Вместо этого в команде указываются регистр базы и смещение. При выполнении машинной команды содержимое поля смещения из команды прибавляется к содержимому регистра базы, в результате получается действительный адрес памяти. В машинных командах, записанных на языке Ассемблера, операндами может указываться неявный адрес, а не значение регистра базы и смещения. Для таких адресов транслятор Ассемблера создает в команде на машинном языке необходимые коды, указывающие значение регистра базы и смещения. Функция транслятора Ассемблера — определять регистр базы и смещение для неявных адресов — дает возможность программисту не представлять каждый адрес в виде этих элементов. Чтобы программист мог воспользоваться этой функцией транслятора, ему предоставлены команды Ассемблера USING и DROP.

3.3.1. Функции оператора USING

Оператор USING (ОПРЕДЕЛИТЬ РЕГИСТР БАЗЫ) предназначен для передачи транслятору Ассемблера информации, необходимой для представления неявного адреса в виде регистра базы и смещения.

Если в исходной программе программист использовал хотя бы один неявный адрес, то оператор USING обязательно должен присутствовать в этой программе. С помощью этого оператора программист сообщает транслятору, каким областям памяти какие общие регистры предназначены в качестве регистров базы, и определяет значения, которые должны находиться в этих регистрах во время выполнения программы, т. е. значения базовых адресов.

Оператор USING только указывает информацию для транслятора Ассемблера. По оператору USING транслятор не строит ни машинной команды, ни константы, поэтому во время выполнения программы по оператору USING никаких действий не произво-

дится. Указанные регистры базы оператор USING не загружает, ответственность за правильную загрузку регистров базы лежит на программисте.

Оператор USING имеет следующий формат:

Название	Операция	Операнды
Имя перехода или пробел	USING	От 2 до 17 выражений в форме: v, r1, r2, ..., r16

Операнд v указывает, что в этом операторе определяются регистры базы для областей памяти со следующими адресами: v, v+4096, v+8192 и т. д. Количество областей зависит от числа операндов (r1, r2 и т. д.), указанных в операторе USING. Операнд v должен быть абсолютным или простым переместимым выражением. Литералы при записи операнда v не допускаются. Переместимое значение операнда v предназначается для неявных переместимых адресов, абсолютное значение — для неявных абсолютных адресов.

Операнды r1, r2, ..., r16 определяют регистры базы и должны быть абсолютными выражениями. Значения этих выражений должны находиться в пределах от 0 до 15. Операнд r1 определяет общий регистр, который может использоваться транслятором как регистр базы и который должен при выполнении программы содержать базовый адрес, равный операнду v. Операнды r2, ..., r16 определяют регистры базы, которые должны содержать базовые адреса, равные v+4096, v+8192, v+12288, ..., v+61440.

Рассмотрим следующий пример:

Название	Операция	Операнды
BEGIN	BALR	7,0
	USING	*7
	L	3,DEB
	A	3,KON
	ST	3,RES
	SVC	14
DEB	DC	F'9'
KON	DC	F'4'
RES	DS	F
	END	

В приведенном примере используются неявные адреса: DEB, KON, RES.

Чтобы транслятор Ассемблера мог представить эти неявные адреса в виде регистра базы и смещения, в программе присутствует оператор USING *, 7. Первый операнд указывает, какой адрес программист использовал в качестве базового адреса, вто-

рой — номер общего регистра, который используется как регистр базы. Знак * в первом операнде оператора USING обозначает ссылку на текущее значение счетчика адреса. Команда BALR имеет длину два байта. Значит, когда транслятор Ассемблера будет обрабатывать оператор USING, значение счетчика адреса будет равно X'2'. Оператор USING сообщает транслятору, что желаемый базовый адрес есть X'2', и общий регистр 7 может быть использован как регистр базы для области памяти, начиная с адреса X'2' до адреса X'1002'. Базовый адрес задается переместимым выражением, поэтому транслятор Ассемблера сможет использовать информацию, сообщенную данным оператором USING, только для представления неявных переместимых адресов, значения которых не выходят из границ области X'2'—X'1001'.

Неявные адреса DEB, KON, RES представляют собой переместимые символические имена, поэтому для представления этих адресов в форме «база — смещение» будет использоваться регистр 7. Если бы в данной программе программист не записал оператор USING, то транслятор не смог бы представить эти неявные адреса в виде регистра базы и смещения и сообщил бы, что операторы, использующие эти адреса, ошибочны.

Рассмотрим, как будет представлена транслятором команда L 3, DEB на машинном языке. Символическое имя DEB получит значение X'10' (2 байта — команда BALR, 4 байта — команда L, 4 байта — команда A, 4 байта — команда ST, 2 байта — команда SVC). Транслятору сообщено оператором USING, что базовый адрес, который должен быть в регистре базы 7, равен X'2'. Значит, транслятор в команду на машинном языке поместит регистр базы 7 и смещение X'0E' (смещение получается вычитанием базового адреса из адреса, определенного символическим именем DEB). Команда на машинном языке будет иметь следующий вид: 58 30 700E.

Общий регистр 0 может определяться оператором USING в качестве регистра базы, но использование регистра 0 для этой цели несколько отличается от использования других регистров. Если общий регистр 0 указывается в операторе USING, то он должен быть записан операндом r1. Операнд v может быть простым переместимым или абсолютным выражением, но в любом случае транслятор Ассемблера предполагает, что регистр 0 содержит значение 0 (переместимое или абсолютное в зависимости от типа выражения, представляющего операнд v). Если в операторе USING, определяющем регистром базы регистр 0, определяются и другие регистры базы, то содержимое следующих по порядку регистров устанавливается равным значениям 4096, 8192 и т. д.

3.3.2. Программирование с оператором USING

Оператор USING может записываться в любом месте программы и столько раз, сколько необходимо для указания регистров базы и содержимого каждого из них. Для каждого неявного

адреса, используемого в программе, должен быть определен доступный регистр базы.

Регистр базы считается доступным для переместимого неявного адреса, если он содержит переместимое значение базового адреса, причем базовый и неявный адреса находятся в одной и той же секции. Регистр базы доступен для абсолютного неявного адреса, если он содержит абсолютное значение базового адреса. При этом регистр базы считается доступным для неявного адреса (абсолютного или переместимого) только тогда, когда он содержит базовый адрес, меньший или равный значению неявного адреса, и разность между значениями неявного и базового адресов, равная смещению, не превышает 4095 байт. В настоящем подразделе рассматривается программирование с оператором USING для одной программной секции. Определение регистров базы в многосекционной программе см. в 3.11.2.

Когда транслятор Ассемблера встречается в какой-либо машинной команде неявный адрес (переместимый или абсолютный), он анализирует имеющуюся информацию о регистрах базы, сообщенную ему программистом с помощью операторов USING к моменту обработки данной команды. Используя эту информацию, транслятор выбирает для данного неявного адреса доступный регистр базы и представляет неявный адрес в форме «база — смещение». Если для неявного адреса существует доступный регистр базы, то говорят, что неявный адрес базируется этим регистром. Процесс определения доступного регистра базы и представления неявного адреса в форме «база — смещение» в дальнейшем часто называется базированием.

Рассмотрим следующий пример:

Название	Операция	Операнды
NAME	BALR USING L	5,Ø *5 3,A
A	DC	F'1Ø'
D	L	4,B
C	DC	F'2Ø'
B	DC END	F'25'

Оператор USING указывает, что общий регистр 5 является регистром базы и содержит базовый адрес, равный X'2'. В программе используются переместимые неявные адреса, определяемые переместимыми символическими именами A и B. Предположим, значение символического имени A равно X'400'. Тогда регистр 5 для неявного переместимого адреса A, используемого в команде с именем NAME, будет доступным: он содержит переме-

стимый базовый адрес, равный $X'2'$; значение базового адреса меньше значения неявного адреса ($X'2' < X'400'$); разность между значениями базового и неявного адресов (смещение) меньше чем 4095 ($X'400' - X'2' = X'3FE'$).

Если и неявный переместимый адрес В отстоит от адреса NAME (значение NAME, равное $X'2'$, загружается в регистр базы) не более чем на 4095 байт, то в данной программе не требуется других операторов USING для определения других регистров базы, так как и для адреса В регистр 5 будет доступен. Если же адрес В отстоит от адреса NAME больше чем на 4095 байт, то регистр 5 нельзя использовать как регистр базы для базирования неявного адреса В. В этом случае оператор D будет отмечен транслятором как ошибочный. Программист в таком случае должен определить в своей программе еще один регистр базы, который был бы доступен для неявного адреса В. Например, можно записать программу в следующем виде:

Название	Операция	Операнды
	BALR	5,Ø
	USING	*5
NAME	L	3,A
A	DC	F'1Ø'
BAS	DC	A(C)
NAME1	L	6,BAS
	USING	C,6
D	L	4,B
C	DC	F'2Ø'
B	DC	F'25'
	END	

В данном случае в программе присутствует второй оператор USING, определяющий еще один регистр базы. Этот оператор указывает транслятору, что регистр базы 6 содержит переместимый базовый адрес С. Команда с именем NAME1 загружает в регистр 6 значение адреса С. Регистр базы 6 будет доступным для неявного адреса В: он содержит переместимый базовый адрес, значение базового адреса (значение имени С) меньше значения неявного адреса (значение имени В), причем только на 4 байта (смещение равно 4). Оператор USING, определяющий регистр базы 6, может быть записан в любом месте программы, но обязательно до команды, использующей неявный адрес В. Если бы программист записал команду определения регистра базы после команды с именем D, то это было бы неправильно. При обработке этой команды, использующей адрес В, транслятор еще не нашел бы доступного регистра для неявного адреса В.

Формат оператора USING позволяет сообщить информацию

сразу о нескольких регистрах базы. Поэтому, если адрес В отстоит от адреса NAME больше чем на 4095 байт, но не больше чем на 8191 байт, в программе можно записать только один следующий оператор: USING *,5, 6. Такой оператор USING сообщит транслятору, что общие регистры 5 и 6 можно использовать в качестве регистров базы и что в момент выполнения программы в регистре 5 должид находиться значение, равное X'2', а в регистре 6 — это значение, увеличенное на 4096. Программист должен только предусмотреть, чтобы эти регистры при выполнении программы были загружены соответствующими значениями.

Как уже отмечалось, оператор USING может появиться в программе несколько раз и в любом месте, как это необходимо для программы. Когда в программе определено несколько регистров базы, может случиться так, что для каких-то неявных адресов будут доступными не один, а несколько регистров. В этом случае при выборе регистров базы транслятор всегда отдает предпочтение доступному регистру базы, дающему минимальную величину смещения. Если имеются два регистра базы с одним и тем же значением базового адреса, то используется регистр с наибольшим номером. Рассмотрим следующий пример:

Название	Операция	Операнды	Идентификация
NAME	BALR	4,Ø	1
	USING	*,4	2
	L	3,FIA	3
	A	3,FIB	4
	L	6,N	5
	USING	NAME,6	6
	ST	3,FIC	7
	L	5,M	8
	L	3,FIA	9
	USING	*,5	1Ø
NAME1	S	3,FIB	11
	ST	3,FID	12
	SVC	14	13
FIA	DC	F'1ØØ'	14
FIB	DC	F'5Ø'	15
FIC	DS	F	16
FID	DS	F	17
N	DC	A(NAME)	18
M	DC	A(NAME1—4)	19
	END		2Ø

В начале программы определен регистр базы 4. Неявные переместимые адреса FIA, FIB, N, используемые в операторах 3, 4 и 5, будут базироваться регистром 4, так как для них будет доступен только этот регистр базы (другие еще не определены): он содержит переместимый базовый адрес, равный значению символического имени NAME (X'2'); этот адрес меньше значений неявных адресов (X'24', X'28', X'34' соответственно); смещение не превышает 4095. Но затем оператор 6 определяет регистром базы

общий регистр 6 и указывает, что базовым адресом является значение символического имени NAME, равное X'2'.

Для неявных адресов FIC, M, FIA будут доступны два регистра базы: 4 и 6. Но так как у этих регистров одинаковое значение базового адреса, то используется регистр с наибольшим номером, т. е. регистр 6. Неявный адрес FIA в операторе 3 будет базироваться регистром 4, а в операторе 9 — регистром 6. В программе присутствует еще оператор 10, определяющий регистр базы 5. Базовый адрес, который указывается этим оператором и загружается в регистр 5 оператором 8, равен значению выражения NAME1—4, т. е. X'1A'.

Для переместимых неявных адресов FIB, FID в операторах 11 и 12 будут доступными три регистра базы: 4, 5, 6. Все эти регистры содержат переместимые базовые адреса, значения базовых адресов (X'2', X'1A', X'2' соответственно) меньше значений неявных адресов FIB, FID (X'28', X'30' соответственно), и разность между значениями адресов FIB, FID и любым базовым адресом меньше 4095. В данном случае для представления неявных адресов FIB и FID в операторах 11 и 12 в качестве регистра базы используется регистр 5, так как этот доступный регистр базы дает минимальное смещение.

Если в программе имеются команды, изменяющие содержимое регистра базы, то необходимо сообщить новое значение базового адреса транслятору с помощью очередного оператора USING. Новый оператор USING отменяет предыдущее значение базового адреса. Транслятор вычисляет смещение, исходя из нового значения базового адреса. В следующей программе, например, регистром базы указан общий регистр 3:

Название	Операция	Операнды	Идентификация
L1	BALR	3,Ø	1
	USING	*3	2
K1	L	3,A	3
	USING	*3	4
	L	5,A1	5
	LA	4,1ØØ	6
A	LA	5,X'1Ø'	7
	SVC	14	8
	DC	A(K1)	9
A1	DC	F'7'	1Ø
	END		11

В операторе 3 для базирования неявного адреса A будет использоваться регистр базы 3, в котором содержится значение X'2' (загружено командой BALR). Оператор 4 опять определяет регистр 3 регистром базы, но с базовым адресом, равным X'6', потому что значение счетчика адреса для оператора 5 равно X'6'. Поэтому в операторе 5 для представления неявного адреса A1 используется регистр базы 3, но смещение вычисляется исходя из нового базового адреса, равного X'6'.

Если бы после второго оператора USING использовались неявные адреса, значения которых меньше нового значения базового адреса X'6', то транслятор не смог бы представить эти адреса в виде регистра базы и смещения, так как для них не нашлось бы доступного регистра базы. Допустим, что после второго оператора USING присутствует команда, использующая неявный адрес L1. Значение этого адреса равно X'2'. В программе в этот момент будет определен только один регистр базы — регистр 3 с базовым адресом, равным X'6'. Это значение больше значения адреса L1, поэтому регистр базы 3 для него недоступен.

Во всех приведенных примерах рассматривались переместимые неявные адреса, для которых оператором USING определялись переместимые базовые адреса из той же самой секции. В программе могут присутствовать адреса, указанные абсолютными выражениями. Например, в предыдущей программе присутствуют команды 6 и 7, вторые операнды которых являются абсолютными адресами. Если в программе присутствуют неявные абсолютные адреса, то программист должен указать в этой программе регистр базы, который должен содержать абсолютный базовый адрес. Значение этого адреса должно быть меньше значения неявного адреса, но разность между их значениями не должна превышать 4095. При таких условиях абсолютные адреса в машинных командах транслятор будет представлять в форме «база — смещение». При определении регистров базы для абсолютных адресов есть исключение. Если значение абсолютного адреса меньше 4096, то для него не требуется определения регистра базы. Значение этого адреса принимается в качестве смещения, а в качестве регистра базы всегда выбирается регистр 0. Например, в предыдущей программе регистр базы с абсолютным базовым адресом не определяется, хотя абсолютные адреса используются. Эти адреса (X'10', 100) меньше 4096, поэтому они будут базироваться регистром 0.

Рассмотрим следующий пример:

Название	Операция	Операнды	Идентификация
A B	BALR	1,0	1
	USING	*,1	2
	L	3,A	3
	LA	4,2000	4
	LA	2,X'1000'	5
	ST	4,X'1020'	6
	USING	X'1000',2	7
	A	3,4	8
	ST	3,X'1020'	9
	LA	4,X'2000'	10
	ST	3,B	11
	SVC	14	12
	DC	F'1000'	13
	DS	F	14
	END		15

В программе в качестве регистров базы определены регистры 1 и 2. Операторы USING указывают, что регистр 1 содержит переместимый базовый адрес, равный текущему значению счетчика адреса $X'2'$, а регистр 2 — абсолютный базовый адрес, равный $X'100'$. Переместимые неявные адреса A и B, используемые в программе, будут базироваться регистром базы 1. В программе используется несколько абсолютных адресов. До оператора USING, определяющего регистр базы 2 и абсолютный базовый адрес, равный $X'100'$, используются следующие неявные абсолютные адреса: 200, $X'100'$, $X'1020'$. Адреса 200 и $X'100'$ меньше 4096, поэтому они будут базироваться регистром 0. Оператор 6 будет отмечен как ошибочный, так как неявный абсолютный адрес $X'1020'$ больше 4095, а регистр базы с абсолютным базовым адресом еще не определен.

В операторе 9 тоже используется неявный абсолютный адрес, который больше 4095. Но оператором 7 уже определен доступный регистр базы для такого неявного адреса — регистр 2. Как указывает оператор USING, регистр 2 содержит абсолютный базовый адрес, равный $X'100'$, который меньше неявного абсолютного адреса ($X'100' < X'1020'$), и разность между ними меньше 4095. В этом случае неявный абсолютный адрес $X'1020'$ будет представлен в виде регистра базы 2 и смещения $X'F20'$. Неявный абсолютный адрес $X'200'$, используемый в операторе 10, также будет представляться с помощью регистра базы 2.

Рассмотрим подробнее случай, когда в качестве регистра базы определен общий регистр 0. Как уже отмечалось, какое бы значение базового адреса ни указывалось в операторе USING, определяющем регистром базы общий регистр 0, транслятор предполагает, что регистр 0 содержит нулевое значение. Смещения для неявных адресов будут вычислены относительно нуля. В зависимости от типа выражения (переместимого или абсолютного), определяющего базовый адрес в операторе USING, транслятор предполагает значение базового адреса, равное 0, переместимым или абсолютным. Исходя из этого транслятор вычисляет смещение относительно 0 для абсолютных или переместимых неявных адресов.

Рассмотрим следующий пример:

Название	Операция	Операнды
A B C	BALR	\emptyset, \emptyset
	USING	$*, \emptyset$
	L	3,A
	A	3,B
	ST	3,C
	LA	$3, X'200'$
	SVC	14
	DC	$F'100'$
	DC	$F'200'$
	DS	F
	END	

Несмотря на то, что оператор USING указывает базовый адрес, равный текущему значению счетчика адреса X'2', транслятор будет использовать для вычисления смещения базовый адрес, значение которого равно 0. Переместимое значение базового адреса в операторе USING позволяет представлять в виде регистра базы и смещения переместимые неявные адреса. Значения неявных адресов A, B, C не превосходят 4096, поэтому в поле смещения будет помещено их значение (смещение относительно 0), а регистром базы будет выбран регистр 0. Значение неявного абсолютного адреса X'200' будет принято в качестве смещения, а в качестве регистра базы — регистр 0, но в этом случае из-за того, что значение неявного абсолютного адреса меньше 4096.

Если в операторе USING, кроме регистра 0, указываются и другие общие регистры, то значения базовых адресов в следующих по порядку регистрах принимаются транслятором равными 4096, 8192 и т. д. независимо от того, какое значение базового адреса указывается оператором USING. Выражение в операторе USING в этом случае только указывает, какие адреса — переместимые или абсолютные — могут быть представлены транслятором в виде регистра базы и смещения с помощью регистров, определяемых оператором USING.

Предположим, в следующем примере длина программы больше 4096 байт, но меньше 8192, и значения символических имен NAME1, NAME2, NAME3 меньше 4096, а значение имен M и N больше или равны 4096.

Название	Операция	Операнды	Название	Операция	Операнды
PERE	USING	PERE, 0, 1	NAME1	DC	F'1000'
	L	4, NAME1	NAME2	DC	F'5'
	A	4, NAME2	NAME3	DS	F
	ST	4, NAME3	.	.	.
	L	5, X'2000'	M	L	3, N
	ST	5, X'12000'	.	.	.
	B	M	N	SVC	14
				DC	F'400'
				END	

В программе определены два регистра базы: 0 и 1. Так как определен регистр базы 0, то транслятор предполагает значения базовых адресов в этих регистрах равными 0 и 4096. Указываемый в операторе USING переместимый базовый адрес PERE не используется транслятором, он только определяет, что в регистрах 0 и 1 находятся переместимые базовые адреса. Каждый из переместимых неявных адресов программы, значения которых меньше 4096 (NAME1, NAME2, NAME3), транслятор представит в виде регистра базы 0 и смещения, равного значению неявного

адреса. Неявные адреса, значения которых равны или больше 4096 (N, M), транслятор представит в виде регистра базы 1 и смещения, равного разности значений неявных адресов и базового адреса 4096. Если бы длина программы была больше 8192 и в ней использовались бы неявные переместимые адреса, значения которых больше 8192, то в операторе USING нужно было бы определить еще столько регистров базы, сколько необходимо.

В приведенной программе используются неявные абсолютные адреса X'200' и X'1200'. Адрес X'200' не требует определения регистра базы: он меньше 4096. Для абсолютного адреса X'1200' должен быть определен доступный регистр базы. В таком виде, как записана программа, этот адрес не может быть представлен с помощью регистра базы и смещения: доступного регистра для этого адреса нет, потому что регистры 0 и 1 содержат базовые адреса, которые являются переместимыми. Если имеющийся оператор USING заменить, например, на оператор USING 0,0,1 или USING X'100',0,1, то транслятор смог бы представить неявный абсолютный адрес X'1200' в виде регистра базы и смещения. Доступным регистром базы был бы регистр 1: оператор USING указывает в этом регистре абсолютный базовый адрес, равный 4096, причем разность между адресом X'1200' и базовым не превышает 4095 ($X'1200' - 4096 = X'200'$). Неявный абсолютный адрес X'1200' транслятор представил бы в виде регистра базы 1 и смещения, равного X'200'. Но при таком операторе USING транслятор не смог бы базировать неявные переместимые адреса.

Оператор USING может записываться в любом месте программы. Программист должен только следить за тем, чтобы операторы USING, определяющие регистры базы, были расположены перед теми командами, которые используют неявные адреса и которые должны использовать определяемые регистры базы. Например, в приведенной ниже программе регистры базы определяются оператором USING не сразу. В начале программы в них нет необходимости: оператор DS определяет область, в команде LA присутствует неявный абсолютный адрес, значение которого меньше 4096 и поэтому для него не требуется определения регистра базы; команда LR не использует неявных адресов. Только перед использованием неявного адреса A регистр 2 определяется как регистр базы. Оператор USING может быть расположен и выше. Например, в начале программы, но только не после оператора b, в котором используется неявный адрес.

В операторе 9 используется неявный адрес NAME. Значение имени NAME меньше, чем значение базового адреса, находящегося в регистре 2. Регистр 2 в этом случае не доступен для имени NAME. Поэтому перед использованием неявного адреса NAME записан еще один оператор USING, определяющий регистр базы 4, доступный для неявного адреса NAME. Этот оператор USING можно разместить в любом месте программы, но обязательно до оператора 9, использующего неявный адрес, который должен базироваться по этому регистру.

Название	Операция	Операнды		Идентификация
NAME	DS	F'5'		1
	LA	6,X'10'		2
	LR	5,6		3
	BALR	2,0		4
	USING	*2		5
NAME1	L	3,A		6
	L	4,B		7
	USING	NAME,4		8
	ST	5,NAME		9
	SVC	14		10
A	DC	F'4'		11
B	DC	A(NAME)		12
	END			13

Если в программе не используются неявные адреса (переместимые или абсолютные, значение которых больше 4095), то программист может не записывать в такой программе оператор USING. Например, в следующей программе оператор USING отсутствует:

Название	Операция	Операнды
	DC	X'5678'
	LA	3,X'20'
	LA	4,X'30'
	AR	3,4
	SVC	14
	END	

3.3.3. Загрузка регистров базы

Как уже отмечалось, оператор USING только сообщает транслятору Ассемблера информацию о регистрах базы и базовых адресах. Транслятор для оператора USING никаких машинных команд и констант не строит, поэтому при выполнении протранслированной программы никаких операций по оператору USING не производится.

При выполнении команды адрес данных вычисляется как сумма содержимого регистра базы и смещения. Но смещение вычислено транслятором относительно базового адреса, указанного оператором USING. Поэтому при выполнении команды базовый адрес, относительно которого вычислено смещение, должен находиться в указанном регистре базы. О правильной и своевременной загрузке всех регистров базы, используемых в программе, обязан позаботиться программист. Он должен в своей программе записать команды, которые при выполнении программы помещают в регистры базы те самые базовые адреса, которые указывались в операторе USING. Команды, устанавливающие содержимое реги-

стров базы, должны выполняться ранее, чем команды, использующие неявные адреса, базируемые этими регистрами базы.

Допустим, имеется программа:

Название	Операция	Операнды
BAS	L	1,A
	L	2,B
	AR	1,2
	ST	1,C
	SVC	14
A	DC	F'1'
B	DC	F'2'
C	DS	F
	END	

Неявные адреса A, B, C должны быть представлены транслятором в виде регистра базы и смещения. Значит, программист должен записать оператор USING, определяющий доступный регистр базы для неявных адресов. Допустим, перед командой с именем BAS записан оператор USING BAS,3. Транслятор представит неявные адреса A, B, C в виде регистра базы 3 и смещения, равного разности между значениями неявных адресов и базового адреса.

При трансляции никаких ошибок не будет обнаружено, потому что транслятор не анализирует, имеются ли в программе машинные команды, которые загружают базовые адреса в регистры базы. В процессе выполнения программы адреса будут вычисляться как сумма содержимого регистра базы 3 и смещения. Но в данной программе отсутствуют машинные команды, которые помещают в регистр 3 необходимый базовый адрес. При выполнении программы регистр 3 будет содержать некоторое произвольное значение, не равное базовому адресу. Поэтому в программе будет осуществляться обращение не к тем областям памяти, к которым необходимо. Для того чтобы программа выполнялась правильно, необходимо загрузить регистр базы 3 значением, указанным в операторе USING. Обычно регистры загружаются с помощью машинной команды L (ЗАГРУЗКА). Но команда L в поле операндов должна содержать адрес памяти. Этот адрес тоже должен быть представлен в виде регистра базы и смещения, а регистр базы еще не подготовлен. Значит, команда L не сможет выполняться правильно. Такая ситуация всегда возникает, когда нужно загрузить базовый адрес в регистр базы в самых первых командах выполняемой программы. Эту задачу можно разрешить, используя команду BALR.

Как можно было заметить в приведенных ранее примерах, при определении регистра базы записывалось сочетание двух операторов: BALR и USING. Учитывая специфику команды BALR, такое сочетание команд дает возможность программисту не толь-

ко определить в программе регистр базы, но и выполнить его загрузку. Операндами команды BALR (ПЕРЕХОД С ВОЗВРАТОМ) являются общие регистры R1 и R2. Действие этой команды состоит в том, что в регистр R1 помещается адрес команды, следующей за командой BALR, а управление передается команде, адрес которой содержится в регистре R2. Если в команде BALR номер регистра R2 равен нулю, то информация в регистре R1 запоминается, но перехода не происходит, а выполняется команда, следующая за BALR. Это свойство команды BALR и обеспечивает возможность загрузки регистра базы в программе. В начале приведенной программы необходимо поместить следующие операторы:

Название	Операция	Операнды
	BALR USING	3,Ø *,3

Команда BALR 3,Ø поместит адрес следующей машинной команды в регистр 3. Следующей выполняемой машинной командой в программе будет команда с именем BAS (оператор USING, следующий за командой BALR, — это команда Ассемблера, которой не соответствует никакая машинная операция). Значит, в регистр 3 будет загружен адрес команды с именем BAS. Оператор USING определяет регистр 3 как регистр базы и сообщает транслятору значение базового адреса в виде знака *, который при размещении USING в этом месте программы соответствует адресу BAS. Таким образом, при трансляции смещение для неявных адресов будет вычисляться относительно адреса BAS, указанного оператором USING, а при выполнении программы именно это значение адреса будет загружено командой BALR в регистр базы. В результате адреса, используемые в программе, будут правильно базироваться при трансляции и вычисляться при выполнении протранслированной программы.

После того как один регистр базы определен, загрузка других регистров базы может быть выполнена с помощью команд загрузки регистров, использующих адреса памяти, хотя команду BALR для этой цели можно использовать в любом месте программы. Команды загрузки регистров базы могут быть расположены в любом месте программы, но они должны выполняться до того, как выполняются команды, где эти регистры базы используются для представления неявных адресов. Например, допустим, что в рассматриваемой ранее программе определяется еще один регистр базы. В программу необходимо будет добавить и команды для загрузки этого регистра.

Программа примет следующий вид:

Название	Операция	Операнды		Идентификация
BAS	BALR	3,Ø		1
	USING	*3		2
	L	1,A		3
	L	2,B		4
	AR	1,2		5
	L	4,D		6
	USING	B,4		7
	ST	1,C		8
	SVC	14		9
	DC	F'1'		10
A	DC	F'2'		11
B	DS	F		12
C	DC	A(B)		13
D	END			14

Транслятор представит неявный адрес C в команде 8 в виде «база — смещение», используя регистр базы 4 и базовый адрес B, так как доступным регистром базы для адреса C будет уже регистр 4, а не 3 (дает меньшую величину смещения). При выполнении программы необходимо, чтобы к моменту выполнения команды 8 регистр 4 был загружен значением B, исходя из которого был пробазирован адрес C.

В программе используется следующий метод загрузки: готовится адресная константа с именем D, равная значению базового адреса, указанному в операторе 7; эта константа загружается в регистр 4 оператором 6. Символическое имя D, используемое в операторе 6, представляет собой неявный адрес. Так как один регистр базы уже определен (регистр 3), то неявный адрес D будет представлен в виде регистра базы 3 и смещения относительно базового адреса BAS. При выполнении программы эта команда выполнится правильно, так как команда 1 загружает в регистр 3 именно значение BAS. Выполнение оператора 6 вызовет загрузку регистра 4 значением, указанным в качестве базового адреса в операторе 7 (значением адреса B). Загрузка регистра базы 4 выполняется раньше, чем команда 8, где регистр 4 используется как регистр базы. При выполнении команды 8 в регистре 4 уже будет находиться базовый адрес для неявного адреса C. Значит, команда 8 будет выполнена правильно.

Название	Операция	Операнды
HERE	BALR	2,Ø
	USING	HERE,2,3,4,5
BASEADDR	LM	3,5,BASEADDR
	B	FIRST
FIRST	DC	A(HERE+4Ø96,HERE+8192,HERE+12288)
	LR	1,12
...	END	

Рассматривался случай, когда в программе один оператор USING определяет несколько регистров базы.

Оператор USING определяет общие регистры 2, 3, 4, 5 регистрами базы. Транслятор считает, что в них содержатся соответственно следующие базовые адреса: HERE, HERE+4096, HERE+8192, HERE+12288. При выполнении программы необходимо, чтобы такие базовые адреса были загружены в соответствующие регистры базы. Регистр базы 2 загружается командой BALR, которая помещает в него адрес следующей за BALR машинной команды (адрес HERE, как и требуется). Для загрузки остальных регистров в программе определяются адресные константы, значения которых равны необходимым базовым адресам. Эти адресные константы определяются оператором DC с именем BASEADDR и будут расположены в памяти друг за другом, занимая три слова. Загрузка адресных констант в регистры выполняется командой групповой загрузки LM, по которой содержимое каждых четырех байт последовательно, начиная с адреса BASEADDR, загружается в регистры 3, 4, 5. Неявный адрес BASEADDR в команде LM будет базироваться регистром 2. При работе программы к моменту выполнения команды с именем HERE, где используется неявный адрес BASEADDR, регистр базы 2 будет уже загружен командой BALR. Значит, команда с именем HERE выполнится правильно, в результате чего будут правильно загружены и остальные регистры базы.

Если в качестве регистра базы используется регистр 0, то каким бы значением регистр 0 не был загружен, оно не влияет на выполнение команд программы, в которых регистр 0 является регистром базы. Если в команде в поле регистра базы стоит нуль, то при вычислении адреса используется нуль, а не то значение, которым загружен регистр 0. Это приводит к тому, что при выполнении такая программа должна всегда располагаться в памяти, начиная с байта 0, и, таким образом, будет непереместимой. Именно поэтому транслятор вычисляет смещение для адресов, базирующихся регистром 0, исходя из того, что в регистре 0 находится нуль, даже если оператор USING указывает базовый адрес, отличный от нуля.

3.3.4. Функции оператора DROP

Команда Ассемблера DROP (ОТМЕНИТЬ РЕГИСТР БАЗЫ) указывает транслятору, что ранее доступный регистр базы не должен больше использоваться в качестве регистра базы.

Оператор DROP имеет следующий формат:

Название	Операция	Операнды
Символическое имя перехода или пробел	DROP	До 16 абсолютных выражений в форме r1,r2,r3,...,r16

Абсолютные выражения указывают общие регистры, которые после оператора DROP становятся недоступными как регистры базы. Значение абсолютных выражений должно находиться в пределах от 0 до 15. Рассмотрим следующий пример:

Название	Операция	Операнды
NAME	BALR	2,0
	USING	*2,3,4
BAS	LM	3,4,BAS
	B	NAME1
NAME1	DC	A(NAME+4096,NAME+8192)
	AR	1,7
...	L	2,BAS+4
	USING	NAME+8192,2
...	DROP	3,4
	END	

В программе оператором USING определяются регистры базы 2, 3, 4. В начале программы выполнена загрузка этих регистров. В дальнейшем в каком-то месте программы оператор DROP запрещает транслятору использовать в качестве регистров базы общие регистры 3 и 4. Начиная с оператора DROP, транслятор не рассматривает регистры 3 и 4 как регистры базы. Оператор DROP можно не использовать, когда базовый адрес регистра изменяется оператором USING, как это делается в приведенной программе для регистра 2.

3.4. ИСПОЛЬЗОВАНИЕ МАШИННЫХ КОМАНД

ЕС ЭВМ располагает большим набором машинных команд, дающих программисту возможность обрабатывать в своей программе различные типы данных. Каждая команда записывается на языке Ассемблера по определенным правилам. Формат записи на языке Ассемблера всех машинных команд ЕС ЭВМ приведен в приложении 2.

3.4.1. Правила записи машинных команд

В поле названия машинных команд может быть записан пробел или любое символическое имя. Под любым символическим именем здесь и в дальнейшем подразумевается один из следующих типов символических имен: простое символическое имя, параметр, символическое имя перехода, соединение параметра со знаками и другими параметрами.

В поле операции машинных команд записывается мнемонический код операции, указывающий операцию, которую должна выполнить ЭВМ по данной машинной команде.

В поле операндов машинных команд записывается один или более операндов. Количество операндов определяется символическим форматом машинной команды. Символические операнды могут изображать регистры, непосредственные операнды и адреса памяти. Операнды, представляющие собой адреса памяти, могут записываться как одно поле или как несколько полей. Например, неявный адрес записывается как одно поле, а явный адрес памяти, записанный в виде регистра базы и смещения, представляет собой операнд, состоящий из поля смещения и следующего за ним поля регистра базы. В командах формата RX в операнде может присутствовать поле регистра индекса, а в командах формата SS — поле длины. Два адреса памяти в командах формата SS представляются независимо друг от друга. Для одного операнда может использоваться неявный адрес и в то же время для другого операнда — явный адрес. Если в командах формата SS требуется указание длины для каждого операнда, то длина каждого операнда представляется независимо: для одного операнда может быть использована неявная длина, в то время как для другого — явная.

Поля в символическом операнде могут быть представлены абсолютными или простыми переместимыми выражениями в зависимости от того, что определяет поле в каждом случае. При составлении выражений должны соблюдаться следующие правила:

- все регистры (общие, с плавающей точкой, базы, индекса) должны определяться абсолютными выражениями, значения которых должны находиться в пределах от 0 до 15;

- смещение при записи явного адреса должно определяться абсолютным выражением, значение этого выражения должно находиться в пределах от 0 до 4095;

- явная длина должна определяться абсолютным выражением; в командах, где требуется указание длины, общей для двух операндов, значение выражения должно быть в пределах от 0 до 256, а в командах, в которых отдельно указывается длина каждого операнда, — в пределах от 0 до 16;

- непосредственный операнд в машинной команде должен определяться абсолютным выражением, значение которого должно находиться в пределах от 0 до 255;

- неявный адрес может записываться в виде абсолютного или простого переместимого выражения.

В поле операндов машинных команд можно записывать комментарии.

При записи поля операндов машинных команд должны соблюдаться следующие правила:

- между полем операндов и полем операции должен быть хотя бы один пробел;

если за операндами записываются комментари, то они от операндов должны отделяться хотя бы одним пробелом;
 операнды в команде должны разделяться запятой;
 для записи нескольких полей в операнде используются скобки (в скобки заключаются регистр базы, регистр индекса и явная длина);

два элемента внутри скобок разделяются запятой;

пробелы в поле операндов допускаются только в знаковом (самоопределенном терме и литерале, определяющем знаковую константу);

если в операнде внутри скобок опускается первое поле, то внутри скобок должна записываться запятая, которая отделяет его от другого поля;

если в операнде внутри скобок опускается второе поле, то запятая, отделяющая его от первого поля, не записывается.

Рассмотрим правила записи машинных команд на примерах. Примеры, которые следуют ниже, объединены по форматам машинных команд. Предполагается, что все символические имена, используемые в примерах, определяются в другом месте в этом же модуле, причем символические имена, которые определяют номера регистров, длину или непосредственный операнд, являются абсолютными выражениями, значения которых находятся в допустимых пределах.

Примеры команд формата RR:

Название	Операция	Операнды	Идентификация
NAME1	LR	2,3	1
NAME2	LR	REG1,REG2	2
NEPD1	SVC	TEN	3

Операнды команды 1 — десятичные самоопределенные термы, которые являются абсолютными значениями. Операнды команд 2 и 3 — символические имена, которые приравниваются в другом месте программы к абсолютным значениям.

Примеры команд формата RX:

Название	Операция	Операнды	Идентификация
NAME1	L	1,5Ø(2,14)	1
NAME2	L	REG1,5Ø(2,SHET)	2
ADR1	L	2,NAD(REG2)	3
ALPHA1	L	2,NAD1	4
BETA1	L	2,-F'1ØØ'	5
BETA2	L	3,5Ø(,1Ø)	6

В командах 1 и 2 используются явные адреса. Регистр базы и регистр индекса записаны в скобках и разделены запятой. REG1 и SHET — абсолютные символические имена. В команде 3 записан неявный адрес, в котором для указания регистра индекса используется абсолютное символическое имя REG2. Имя NAD может быть абсолютным или переместимым символическим именем. Регистр базы при записи неявного адреса не указывается, запятая после указания регистра индекса в случае отсутствия регистра базы не записывается, регистр индекса записывается в скобках. В команде 4 записан неявный адрес, регистр индекса не используется, операнд записан как одно поле. Символическое имя NAD1 может быть абсолютным или переместимым. В команде 5 второй операнд является литералом. В команде 6 записан явный адрес, но регистр индекса не используется. В этом случае запятая, отделяющая его от регистра базы, записывается.

Примеры команд формата RS:

Название	Операция	Операнды	Идентификация
NAME1	BXH	1,2,4Ø(12)	1
NAME3	BXH	REG1,REG2,FIELD	2
SDB1	SLL	REG5,12	3

В команде 1 определяется явный адрес. Все регистры и смещение записаны в виде десятичных самоопределенных термов. Команда 2 использует неявный адрес, имя FIELD может быть абсолютным или переместимым. Символические имена REG1 и REG2, обозначающие регистры, должны быть абсолютными. В команде 3 записан неявный адрес в виде абсолютного выражения. Имя REG5 в этой команде должно быть абсолютным.

Примеры команд формата SI:

Название	Операция	Операнды	Идентификация
NAME1	CLI	4Ø(REG1Ø),X'4Ø'	1
ADR1	CLI	NEV,SOR	2
GAM1	SIO	4Ø(9)	3
GAM2	SIO	NEV	4

В командах 1 и 3 записаны явные адреса. Команды 2 и 4 используют неявные адреса. При записи явных адресов используются абсолютные выражения. Непосредственный операнд, указываемый в командах формата SI, записывается в символическом формате за адресом памяти, а не перед ним, как он представля-

ется в машинном формате. Непосредственный операнд задается абсолютным выражением.

Примеры команд формата SS:

Название	Операция	Операнды	Идентификация
NAME1	AP	4Ø(NINE,REG8),TRID(6,7)	1
NAME2	AP	FID2,FID1	2
ADR1	AP	FID2(1Ø),FID1(5)	3
ADR3	AP	4Ø(NINE,8),TRID(,7)	4
ADR4	AP	TRID(,REG8),FID1(5)	5
GAM1	MVC	4Ø(9,8),3Ø(7)	6
GAM2	MVC	4Ø(,REG8),TRID(7)	7
GAM3	MVC	FID2,FID1	8
GAM4	MVC	FID2(9),FID1	9

В команде 1 записаны явные адреса и явные длины. Длина и регистр базы в первом операнде записаны в виде символических имен, символические имена NINE, REG8 и TRID должны быть абсолютными:

В команде 2 оба адреса записаны в неявном виде, и для них используется неявная длина.

В команде 3 записаны неявные адреса и указана явная длина для каждого операнда. При отсутствии регистра базы в скобках записывается только длина. Запятая, отделяющая ее от регистра базы, не записывается. Символические имена FID1 и FID2 могут быть абсолютными или переместимыми.

В команде 4 оба адреса явные, но в одном указана явная длина, во втором — неявная. При записи явного адреса, где указана явная длина, в скобках записана запятая, отделяющая длину от регистра базы.

В команде 5 первым записан явный адрес, при этом используется неявная длина, что указано записанной в скобках запятой, а вторым — неявный адрес, но указана явная длина. При записи неявного адреса регистр базы не записывается, поэтому в скобках записана только длина, запятая после нее не записывается. Во всех рассматриваемых ранее командах формата SS требовалось указание длины для обоих операндов. Значение длины (явной или неявной) должно было находиться в пределах от 0 до 16.

В командах 6, 7, 8 и 9 длину необходимо указывать только в первом операнде. Значение длины в этом случае может находиться в пределах от 0 до 256. В команде 6 записаны оба явные адреса и указана явная длина, в команде 7 тоже записаны явные адреса, но используется неявная длина. Регистр базы первого адреса и смещение во втором адресе в команде 7 записаны в виде символического имени, символические имена REG8 и TRID должны быть абсолютными. В командах 8 и 9 оба адреса неявные, но в первом случае используется неявная длина, а во втором — указана явная длина. Длина записывается в скобках.

3.4.2. Команды с фиксированной точкой

Команды с фиксированной точкой выполняют операции над числами с фиксированной точкой длиной в слово или полуслово. Эти команды используют общие регистры и могут быть следующих форматов: RR, RX и RS. Команды с фиксированной точкой выполняют следующие операции: загрузка, сложение, вычитание, умножение, деление, запись в память, перевод чисел из десятичной системы в двоичную и из двоичной системы в десятичную, сдвиг. Операнды в командах с фиксированной точкой должны быть расположены на границе слова или полуслова (в зависимости от длины операндов). Транслятор Ассемблера проверяет, находятся ли операнды в памяти на нужной границе.

При выполнении большинства команд с фиксированной точкой устанавливается признак результата. Признак результата может быть использован для выбора пути при выполнении последующих команд условного перехода. Признак результата устанавливается равным одному из четырех значений: 0, 1, 2 или 3. Для большинства операций признак результата 0, 1 или 2 указывает, что результат операции соответственно равен нулю, меньше или больше нуля. Признак результата, равный 3, указывает на то, что произошло переполнение. В операциях сравнения признак результата 0, 1, 2 указывает, что первый операнд соответственно равен второму, меньше или больше его.

При выполнении операций с фиксированной точкой в некоторых случаях возникает программное прерывание, по которому выполнение программы прекращается. Прерывания программы вызывают следующие причины: защита памяти, адресация, спецификация, данные, переполнение, некорректность деления с фиксированной точкой.

Прерывание по защите памяти возникает в том случае, если ключ памяти для операнда не совпадает с ключом защиты в слове состояния программы. Это происходит в том случае, когда команда обращается к байту памяти, находящемуся вне той области памяти, которая выделена для выполнения программы.

Прерывание по адресации происходит в том случае, если адрес, указанный в команде, превышает допустимый для данной установки. Например, прерывание по адресации произойдет, если команда обращается к байту 70000, а объем памяти на данной ЭВМ составляет только 65536 байт.

Прерывание по спецификации возникает, если операнд длиной в слово или полуслово не начинается в основной памяти с соответствующей границы слова или полуслова. Кроме того, прерывание по спецификации происходит также в том случае, если в команде указан нечетный регистр для пары общих регистров, содержащих операнд длиной 64 разряда.

Прерывание по данным вызывает неправильный код знака или цифры десятичного операнда при выполнении команды CVB (ПРЕОБРАЗОВАНИЕ В ДВОИЧНУЮ).

Прерывание по переполнению происходит в том случае, если результат выполнения операции превышает допустимый диапазон данных. Это прерывание можно замаскировать в слове состояния программы.

Некорректность деления с фиксированной точкой возникает в следующих случаях: частное превышает размер регистра; происходит деление на нуль; результат выполнения команды CVB превышает 31 разряд.

Рассмотрим использование некоторых команд с фиксированной точкой на конкретных примерах. Команды сдвига будут рассмотрены вместе с логическими командами, команды сравнения и некоторые команды загрузки — вместе с командами переходов.

Сложение и вычитание. Допустим, необходимо подсчитать количество рабочих на предприятии в конце месяца, если известно количество рабочих в начале месяца и количество рабочих, которые были приняты на работу и уволены в течение месяца. Эти вычисления можно выполнить, используя следующую формулу:

$$KONM = HASCHM + PRIB - VIB,$$

где HASCHM — количество рабочих в начале месяца,

PRIB — количество прибывших рабочих,

VIB — количество выбывших рабочих.

Предположим, что HASCHM=100, PRIB=12, VIB=5.

Тогда программа, выполняющая подсчет по формуле, будет следующей:

Название	Операция	Операнды
BEGIN	BALR USING L A S ST SVC	4,0 *,4 5,HASCHM 5,PRIB 5,VIB 5,KONM 14
HASCHM	DC	F'100'
PRIB	DC	F'12'
VIB	DC	F'5'
KONM	DS END	F BEGIN

Операторы с именами HASCHM, PRIB, VIB определяют исходные данные в виде чисел с фиксированной точкой, каждое из которых имеет длину 4 байта (одно слово). Оператор DS с именем KONM определяет область памяти для результата длиной в 4 байта. Операторы DC и DS описаны в 3.6 и 3.8.

Первые операторы BALR и USING загружают и определяют регистр базы. Далее следует первая команда обработки, которую рассмотрим подробнее. Команда L (ЗАГРУЗКА) является командой формата RX, длина ее равна четырем байтам. По этой

команде выбираются четыре байта из области памяти, адрес которой указывается вторым операндом, и помещаются в общий регистр, указанный первым операндом. Адрес, указываемый в команде, должен начинаться с границы слова, т. е. должен быть кратен 4. В команде, записанной в программе, буква L — мнемонический код операции, 5 — номер общего регистра, в который будет загружаться содержимое области памяти, НАСНМ — символическое имя слова основной памяти, которое необходимо загрузить в регистр 5. Транслятор представит этот адрес в форме «база — смещение».

Команда A (СЛОЖЕНИЕ) также является командой формата RX. Она выполняет сложение двух чисел с фиксированной точкой. Одно число находится в регистре, другое — в основной памяти. По этой команде содержимое слова основной памяти складывается с содержимым общего регистра, результат операции помещается в регистр. Номер общего регистра указывается в команде первым операндом, адрес основной памяти — вторым. В команде A, записанной в программе, указан общий регистр 5 и адрес PRIB. В регистр 5 предыдущей командой L загружено число, расположенное по адресу НАСНМ. По команде A к этому числу будет прибавлено число, расположенное по адресу PRIB.

— По следующей команде S (ВЫЧИТАНИЕ) происходит вычитание числа, расположенного по адресу VIB, из числа, находящегося в регистре 5. Формат команды S такой же, как и команд L, A. Результат операции, являющийся результатом вычислений по формуле, помещается в тот же регистр 5. По условию его необходимо поместить в область памяти KONM. Запись содержимого общего регистра в память выполняется командой ST (ЗАПИСЬ В ПАМЯТЬ). Содержимое общего регистра при выполнении этой команды не меняется.

Рассмотренные в этом примере команды L, A, S, ST имеют формат RX. Они позволяют использовать регистр индекса, но в данном примере в этом нет необходимости.

После выполнения вычислений в программе записана команда SVC. Эта команда используется в тех случаях, когда нужно выполнить какое-нибудь действие с помощью операционной системы, под управлением которой выполняются все проблемные программы. В наших примерах будем использовать команду SVC с операндом 14. Этот операнд указывает на прекращение работы программы.

Оператор END сообщает транслятору о том, что программа закончена, и указывает, что первой командой, с которой должно начаться выполнение программы, является команда с именем BEGIN.

Команды L, A, S, ST обращаются к областям памяти длиной в слово. Для выполнения соответствующих операций над полусловами используются команды LH, AH, SH и STH.

Данные можно было поместить сначала в общие регистры, а затем для сложения и вычитания можно было использовать

команды AR и SR формата RR, выполняющие действия над содержимым двух регистров. В этом случае реализовать вычисления по приведенной формуле можно следующими командами:

Название	Операция	Операнды
	L	5,HACHM
	L	6,PRIB
	AR	5,6
	L	6,VIB
	SR	5,6
	ST	5,KONM

Если числа с фиксированной точкой длиной в слово расположены в памяти друг за другом и их нужно поместить в регистры, номера которых последовательны, то можно использовать команду LM (ЗАГРУЗКА ГРУППОВАЯ). Аналогично для сохранения в памяти содержимого регистров, номера которых последовательны, можно использовать команду STM (ЗАПИСЬ В ПАМЯТЬ ГРУППОВАЯ). Команды LM и STM — команды формата RS. Адрес, записываемый в этих командах, не индексируется. В поле операндов этих команд указываются номера двух общих регистров и адрес области памяти. По команде LM в общие регистры, начиная с регистра с номером, указанным в команде первым, и кончая регистром с номером, указанным вторым, загружаются данные из области памяти, адрес которой указан в команде. Первое слово из области попадает в первый указанный регистр, второе — в регистр с номером на 1 больше и т. д. Общие регистры загружаются в порядке возрастания их номеров до тех пор, пока не будет загружен регистр с номером, указанным в команде последним. При этом считается, что за регистром с номером 15 следует регистр с номером 0. В каждый последующий регистр загружается содержимое последующего слова из области памяти. Аналогично действуют команды STM, по которой содержимое группы общих регистров последовательно сохраняется в указанной области памяти. Используя команду LM, вычисления по формуле можно реализовать следующим образом:

Название	Операция	Операнды
	LM	5,7,HACHM
	AR	5,6
	SR	5,7
	ST	5,KONM

По команде LM в регистр 5 загрузится число, расположенное по адресу HACHM, занимающее 4 байта, в регистр 6 — расположенное по адресу PRIB, так как оно находится в следующих че-

тырех байтах, в регистр 7 — число, расположенное по адресу VIB.

Умножение и деление. В качестве простого примера на умножение рассмотрим следующую задачу. Допустим, необходимо получить общую стоимость выпущенной продукции, если известно, что выпущено N единиц продукции, а цена каждой единицы равна K . Стоимость (STOIM) подсчитывается по формуле $STOIM = N \cdot K$. Предположим, что $N=10$, $K=25$. Программа, реализующая это вычисление, будет следующей:

Название	Операция	Операнды
BEGIN	BALR	5,0
	USING	*,5
	L	11,N
	M	10,K
	ST	11,STOIM
	SVC	14
N	DC	F'10'
K	DC	F'25'
STOIM	DS	F
	END	BEGIN

Исходные данные в программе определены операторами DC с именами N и K в виде чисел с фиксированной точкой. Оператором DS с именем STOIM резервируется область для результата вычислений.

Первые два оператора программы BALR и USING выполняют загрузку и определение регистра базы. Команда L (ЗАГРУЗКА) помещает число, расположенное по адресу N (количество единиц продукции), в общий регистр 11.

Команда M (УМНОЖЕНИЕ) выполняет умножение двух чисел с фиксированной точкой. Это команда формата RX. По этой команде содержимое общего регистра умножается на содержимое слова основной памяти. Результат умножения всегда будет больше любого из сомножителей и может иметь длину до 64 бит. Поэтому результат умножения помещается в пару общих регистров: в регистр, указанный в команде, и в регистр с номером на 1 больше. Требуется, чтобы в команде в качестве первого операнда всегда был указан регистр с четным номером. Если будет указан регистр с нечетным номером, то это будет рассматриваться как ошибка. Множимое должно находиться в нечетном регистре, номер которого на 1 больше регистра, указанного в команде. Таким образом, по команде M, записанной в программе, содержимое регистра 11 (количество единиц выпущенной продукции) будет умножено на содержимое слова с адресом K (цена единицы продукции). Для результата предназначены общие регистры с номерами 10 и 11.

После того как произведение получено, результат запоминается в области STOIM. Предполагается, что длина результата не превышает длину одного регистра. За обоснованность такого допущения отвечает программист. В противном случае программист должен обеспечить запись в память обеих частей произведения.

Рассмотрим следующий пример, использующий команду деления. Задача состоит в том, чтобы подсчитать общую стоимость продукции, если известно, что она увеличена по сравнению с известной ранее на 3%. Допустим, начальная общая стоимость продукции была 2489. Программа, реализующая поставленную задачу, будет выглядеть следующим образом:

Название	Операция	Операнды
BEGIN	BALR	15,0
	USING	*,16
	L	5,STOIM
	M	4,YVEL
	D	4,C100
	ST	5,RESULT
	SVC	14
STOIM	DC	F'2489'
YVEL	DC	F'103'
C100	DC	F'100'
RESULT	DS	F
	END	BEGIN

В программе после обычных предварительных операций в регистр 5 из области STOIM загружается начальное значение общей стоимости. Пара регистров 4 и 5 предназначена для умножения. Множимое помещается в регистр 5. Коэффициент увеличения продукции изображается числом 103, находящимся в области YVEL, что должно восприниматься как 1,03. Это сделано в целях сокращения программы; вместо того чтобы умножать размер стоимости на 0,03 и прибавлять полученное произведение к величине начальной стоимости, начальная стоимость умножается на 1,03. Результат получается такой же, причем в последнем случае нет необходимости выполнять сложение. Предварительная стоимость умножается на 103 вместо 1,03, в программе произведение будет в 100 раз больше правильного результата. Поэтому после выполнения умножения выполняется деление на 100, чтобы результат был верный. Считаем, что в любом случае результат умножения можно поместить в один регистр 5.

Команда D (ДЕЛЕНИЕ) выполняет деление чисел с фиксированной точкой. Делимое должно находиться в паре четного-нечетного регистров как 64-разрядное число. Если в команде D используется регистр с нечетным номером, то это рассматривается как ошибка. Остаток от деления помещается в четный регистр,

а частное — в нечетный регистр. В данном случае делимое находится в паре регистров 4 и 5, частное будет помещено в регистр 5, остаток — в регистр 4. Остаток для программы безразличен. Частное сохраняется в области RESULT.

Умножение можно производить и над числами с фиксированной точкой длиной в полуслово, применяя команду MN. Деление выполняется только над числами длиной в слово. Для умножения и деления могут быть использованы и команды формата RR (MR — УМНОЖЕНИЕ, DR — ДЕЛЕНИЕ). В этом случае множитель (или делитель) содержится в общем регистре, а не в области памяти, но все требования для множимого (или делимого) остаются прежними.

Перевод чисел. Преобразование чисел из двоичной системы счисления в десятичную и, наоборот, в ЕС ЭВМ можно выполнить с помощью команд CVB (ПРЕОБРАЗОВАНИЕ В ДВОИЧНУЮ) и CVD (ПРЕОБРАЗОВАНИЕ В ДЕСЯТИЧНУЮ).

Допустим, в результате каких-то действий над данными в памяти получено число 123 в упакованном десятичном формате. Необходимо к числу 123 добавить число 100, причем число 100 определено как число с фиксированной точкой, т. е. в двоичном формате; полученный результат необходимо сохранить в упакованном десятичном формате. Сложение можно выполнить, применив команду A (СЛОЖЕНИЕ). Но эта команда складывает числа с фиксированной точкой. Значит, десятичное число необходимо предварительно преобразовать в двоичную систему счисления. Рассмотрим реализацию этой задачи в программе.

Название	Операция	Операнды
BEGIN	BALR	15,Ø
	USING	*J15
	CVB	5,DEC
	A	5,DV
	CVD	5,RES
	SVC	14
RES	DS	D
DEC	DC	PL8'123'
DV	DC	F'1ØØ'
	END	BEGIN

Команда CVB является командой формата RX (в приведенном примере регистр индекса не используется). По команде CVB число, находящееся в области памяти, адрес которой указан вторым операндом команды, преобразуется из десятичной системы счисления в двоичную, и результат помещается в регистр, указанный первым операндом. Как до преобразования, так и после него число рассматривается как целое число со знаком. Десятичное число должно быть представлено в упакованном формате,

занимать в памяти двойное слово и располагаться на границе двойного слова. Четыре младших разряда двойного слова используются для представления знака, остальные 60 разрядов содержат 15 цифр в двоично-десятичном представлении. При выполнении команды десятичный операнд проверяется на правильность кодов знака и цифр. Наличие неправильных кодов вызывает прерывание программы. Наибольшее десятичное число, которое может быть преобразовано с помощью команды CVB и помещено в общий регистр, равно 2147483647, наименьшее — равно —2147483648. Отрицательные числа после перевода представляются в дополнительном коде.

В приведенной программе оператор DC с именем DEC определяет десятичное число в упакованном формате длиной 8 байт (определение десятичных чисел оператором DC описано в 3.6.2.). Число будет расположено на границе двойного слова, так как предыдущий оператор DS с именем RES устанавливает счетчик адреса на нужную границу (о возможностях выполнения выравнивания с помощью оператора DS см. в 3.8.2). По команде CVB десятичное число 123, расположенное по адресу DEC, преобразуется в двоичную систему, и результат помещается в регистр 5. Содержимое регистра 5 после выполнения команды CVB будет: X'0000007B'. Если бы число было отрицательным, т. е. —123, то в результате выполнения команды CVB содержимое регистра 5 было бы следующим: X'FFFFFF85'.

Команда A выполняет сложение числа с фиксированной точкой, определенного оператором DC с именем DV, с числом, находящимся в регистре 5. Результат сложения (X'000000DF') помещается в регистр 5. Полученное двоичное число необходимо преобразовать в десятичный упакованный формат. Это выполняется с помощью команды CVD.

Команда CVD является командой формата RX. По команде CVD содержимое общего регистра, указанного первым операндом, преобразуется из двоичной системы счисления в десятичную, а результат помещается в область памяти, адрес которой указан вторым операндом. Как до преобразования, так и после число рассматривается как целое число со знаком. Отрицательные числа должны быть представлены в дополнительном коде. Результат представляет собой двойное слово и должен быть помещен в память, начиная с границы двойного слова. Младшие четыре разряда результата используются для представления знака: плюс кодируется комбинацией 1100, минус — 1101. Остальные 60 разрядов содержат 15 цифр в двоично-десятичном коде. Переполнения при выполнении этой команды никогда не возникает, потому что результат всегда помещается в отведенную область. В приведенной программе по команде CVD преобразуется содержимое регистра 5. Результат сохраняется в области RES и представляется в памяти следующим образом: первые 48 разрядов двойного слова — нули, последние 16 — 0010001000111100.

3.4.3. Десятичные команды

Набор команд десятичной арифметики в ЕС ЭВМ обеспечивает возможность выполнения арифметических операций над десятичными числами (сложение, вычитание, умножение, деление), а также операций сравнения и преобразования формата данных. Все десятичные команды являются командами формата SS. В каждой команде используются два адреса основной памяти. Команды обрабатывают данные переменной длины. Адрес всегда задается самым левым байтом области, в которой находится операнд. Результат операций помещается в область, занимаемую первым операндом команды.

Десятичные команды обрабатывают только данные в упакованном десятичном формате. Единственное исключение представляет команда РАСК (УПАКОВАТЬ), у которой один операнд должен быть десятичным числом с зоной. Операнды могут начинаться с любого байта и иметь длину от 0 до 16 байт. В десятичных командах длина указывается отдельно для каждого операнда. Эта длина может быть одинаковой или разной для двух операндов. В последнем случае при выполнении команды более короткий операнд дополняется нулями слева от старшей цифры. Результаты никогда не выходят за пределы, установленные адресом и длиной. Длина в команде на машинном языке может принимать значения от 0 до 15. При записи оператора на языке Ассемблера программист указывает истинную длину операнда, а ее значение в машинной команде, уменьшенное на 1, устанавливает транслятор (длину 0 транслятор на 1 не уменьшает).

Во всех командах десятичной арифметики, за исключением команд РАСК, UNPK (РАСПАКОВАТЬ) и MVO (ПЕРЕСЫЛКА СО СДВИГОМ), операнды либо совсем не должны перекрываться, либо у них должны совпадать самые правые байты. В команде ZAP (СЛОЖЕНИЕ С ОЧИСТКОЙ) поле результата может перекрываться с полем исходного данного, но при этом самый правый байт первого операнда должен находиться правее самого правого байта второго операнда или совпадать с ним.

При выполнении операций сложения, вычитания и сравнения устанавливается признак результата. Для операции сравнения признак результата 0, 1, 2 указывает на то, что первый из операндов соответственно равен, меньше или больше второго. Для остальных операций признак результата 0, 1, 2 или 3 устанавливается в тех случаях, когда результат операции соответственно равен нулю, меньше нуля, больше нуля или произошло переполнение.

При выполнении операций десятичной арифметики могут вызвать прерывание следующие причины: защита памяти, адресация, спецификация, данные, десятичное переполнение и десятичное деление.

Прерывание по защите памяти происходит в том случае, если

ключ памяти операнда не совпадает с ключом защиты в слове состояния программы.

Прерывание по адресации возникает, если программа обращается к байту памяти, отсутствующему в данной ЭВМ.

Прерывание по спецификации происходит, если длина множителя или делителя больше, чем 15 цифр плюс знак или больше, чем длина множимого или делимого.

Если команда десятичной арифметики пытается выполнить операцию не над десятичными числами, если в старших разрядах множимого недостаточное количество нулей или если поля операндов неправильно перекрываются, то возникает прерывание по данным.

Переполнение результата, возникающее при выполнении операций сложения и вычитания над десятичными числами, вызывает прерывание по десятичному переполнению. Это прерывание можно замаскировать в слове состояния программы.

Прерывание по десятичному делению возникает в том случае, если частное не помещается в отведенное для него поле.

Десятичные данные можно обрабатывать также с помощью логических операций (например, операций пересылки, логического сравнения, редактирования и т. д.), которые рассматриваются в 3.4.5.

Рассмотрим использование команд, реализующих операции над десятичными данными, на конкретных примерах.

Сложение и вычитание в десятичной арифметике. Рассмотрим пример, приведенный при разборе команд с фиксированной точкой, и запишем его с использованием команд десятичной арифметики. По условию известно количество рабочих в начале месяца (HACHM) и количество рабочих, принятых и уволенных в течение месяца (PRIB и VIB). Необходимо подсчитать количество рабочих в конце месяца. Предположим, что все исходные данные представлены в упакованном формате. Реализовать нужные вычисления можно с помощью следующей программы:

Название	Операция	Операнды
BEGIN	BALR	15,0
	USING	*,15
	MVC	KONM,HACHM
	AP	KONM,PRIB
	SP	KONM,VIB(4)
	SVC	14
HACHM	DC	PL4'1000'
PRIB	DC	PL4'12'
VIB	DC	PL4'5'
KONM	DS	PL4
	END	BEGIN

Операторы DC определяют десятичные упакованные числа длиной 4 байта. Необходимые арифметические операции в программе будут выполняться прямо над десятичными числами.

В начале программы записаны операторы BALR, USING, определяющие регистр базы.

Первая команда обработки — команда MVC (ПЕРЕСЫЛКА) — относится к группе команд, выполняющих логические операции, которые можно использовать при работе с десятичными данными. Команда MVC — команда формата SS — помещает второй операнд на место первого. Операнды могут любым способом перекрываться, пересылка идет слева направо байт за байтом. В команде указывается общая длина для двух операндов, потому что область, откуда выбираются данные, и область, куда они помещаются, должны иметь одну и ту же длину. Длина может быть задана в пределах от 0 до 256 байт. В данном случае в команде MVC используется неявная длина символического имени KONM, которая равна 4. Таким образом, по команде MVC четыре байта, начиная с адреса HACHM, пересылаются в область, начиная с адреса KONM, т. е. константа с именем HACHM помещается в область KONM. Это сделано в связи с тем, что программы рекомендуется составлять таким образом, чтобы они не изменяли исходных данных. Десятичные команды не используют общих регистров, поэтому для выполнения этой рекомендации необходимо выделять области памяти, в которых будут проводиться все вычисления. В данном случае такой областью является область KONM.

По команде AP (СЛОЖЕНИЕ ДЕСЯТИЧНОЕ) к десятичному числу в области KONM (начальное количество рабочих) добавляется число из области PRIB (число прибывших рабочих). Результат помещается на место первого операнда в область KONM. При сложении выполняется проверка на правильность кодов знаков и цифр. Если в области, предназначенной для результата, не помещаются все значащие цифры суммы, возникает десятичное переполнение.

По команде SP (ВЫЧИТАНИЕ ДЕСЯТИЧНОЕ) из суммы, находящейся в области KONM, вычитается десятичное число из области VIB (число выбывших рабочих). Результат команды SP помещается на место первого операнда в область KONM. Выполнение команды SP аналогично выполнению команды AP. Отличие состоит лишь в том, что знак второго операнда перед выполнением операции изменяется на противоположный.

Десятичное умножение. Десятичное умножение рассмотрим на уже знакомой задаче подсчета выпускаемой продукции, когда известна величина выпускаемой продукции в начале года и коэффициент увеличения выпуска продукции за год. Программа, реализующая необходимые вычисления, будет следующей:

Название	Операция	Операнды
BEGIN1	BALR USING ZAP MP MVN MVC SVC	15,0 *,15 RES,PROD RES, YVEL RES+4(1),RES+5 RESULT,RES+1 14
PROD	DC	PL4'2489'
YVEL	DC	PL2'103'
RES	DS	PL6
RESULT	DS	PL4
	END	BEGIN1

Исходные данные определены с помощью операторов DC в виде упакованных десятичных чисел: величина начальной продукции — константа с именем PROD длиной 4 байта, коэффициент увеличения — константа с именем YVEL длиной 2 байта. Для результата предназначена область RESULT.

Вычисления выполняются с помощью команды MP (УМНОЖЕНИЕ ДЕСЯТИЧНОЕ). В этой команде второй операнд является множителем, область первого операнда вначале содержит множимое, а после завершения операции — произведение. Длина множителя не должна превышать 7 байт и должна быть меньше длины множимого. Так как число цифр в произведении равно сумме цифр операндов, в множимом должно быть по крайней мере столько нулевых старших цифр, сколько цифр имеет множитель. Знак в произведении определяется по правилам алгебры, даже если один или оба операнда нулевые.

В приведенной программе необходимо умножить десятичное число с именем YVEL на десятичное число с именем PROD. В команде MP нельзя указывать число с именем YVEL как множимое, так как его длина будет меньше длины числа с именем PROD, указанного в этом случае множителем. Нельзя сразу же выполнить и команду MP, в которой было бы указано как множимое число с именем PROD, так как это число содержит только три нулевые цифры слева, тогда как у множителя (числа с именем YVEL) насчитывается четыре цифры. Поэтому число с именем PROD пересылается в рабочую область памяти, в старших разрядах которой предусматриваются дополнительные нули. Это выполняется командой ZAP (СЛОЖЕНИЕ С ОЧИСТКОЙ). По этой команде в область, адрес которой указан первым операндом (в данном случае RES), записываются нули, а затем к первому операнду прибавляется второй операнд (PROD). Область RES на два байта больше PROD.

Таким образом, после выполнения команды ZAP в области RES будут находиться по крайней мере четыре десятичных нуля (два нулевых байта). При выполнении команды ZAP ведется

контроль правильности кодов знаков и цифр. Если в области, адрес которой указан первым операндом, не помещаются все значащие цифры второго операнда, возникает десятичное переполнение.

После команды ZAP будет выполняться команда десятичного умножения. При выбранных для примера значениях результат в области RES будет равен числу 00000256367C (C — знак плюс). Исходное число необходимо было умножить на число 1,03, а не на 103, поэтому произведение содержит два десятичных знака после запятой и равно 2563,67. Предположим, что результат должен быть целым числом. В этом случае необходимо отбросить в результате две цифры справа.

В данном случае не будем делать округление, которое можно выполнить, сложив результат с десятичной константой 50C. Невозможно убрать две последние десятичные цифры из результата простой пересылкой целой части числа в область RESULT, потому что при такой пересылке потеряется знак числа. Поэтому прежде чем переслать целую часть числа в область RESULT, необходимо переслать туда знак результата. В данном примере для работы с десятичными данными используется еще одна команда из группы логических команд — команда MVN (ПЕРЕСЫЛКА ЦИФР).

Команда MVN — команда формата SS. По этой команде четыре младших разряда (цифра) каждого байта из области, адрес которой указан вторым операндом, помещаются в младшие разряды соответствующих байт области, адрес которой указан первым операндом. Пересылка идет слева направо байт за байтом. Поля могут перекрываться любым образом. Цифры пересылаются без изменения. Проверка, является ли данная комбинация цифрой, не делается. Старшие четыре разряда каждого байта остаются неизменными в обоих операндах.

В приведенном примере по команде MVN четыре последних разряда байта с адресом RES+5 (последнего байта результата вычислений, содержащего цифру 7 и знак) помещаются в байт с адресом RES+4, т. е. знак будет помещен на место цифры 6, которую нужно отбросить. Пересылаются младшие разряды только одного байта, потому что в команде указана явная длина операнда, равная 1. Команда MVN использует длину только первого операнда. В результате выполнения этой команды содержимое области RES будет 000002563C7C. Теперь результат можно поместить в область RESULT. Результат вычислений находится в области RES и имеет длину 6 байт. Командой MVC в область RESULT пересылаются 4 байта, начиная с адреса RES+1. Неявная длина имени RESULT равна 4. Поэтому нет необходимости указывать в команде явную длину. В пересылке не участвуют крайний левый байт, содержащий два нуля, и крайний правый байт области RES, содержащий цифру и знак.

Десятичное деление. Сначала рассмотрим, как выполняется команда десятичного деления.

Команда DP (ДЕСЯТИЧНОЕ ДЕЛЕНИЕ) является командой формата SS. Первый операнд — делимое, второй — делитель. Результат операции, состоящий из частного и остатка, помещается на место делимого. Частное помещается в левую часть области, остаток — в правую. Остаток имеет ту же длину, что и делитель. Частное и остаток занимают всю область делимого, следовательно, адресом частного является адрес первого операнда. Делимое, делитель, частное и остаток являются целыми числами со знаками. Знак частного определяется по правилам алгебры в соответствии со знаками делимого и делителя. Знак остатка тот же, что и у делимого. Поля делителя и делимого могут перекрываться, но так, чтобы совпадали их младшие байты.

Предположим теперь, что необходимо разделить число 4246 на 31. В области DELIM длиной 5 байт расположим число 4246 (000004246C), а в области DELIT — число 31 (031C). Область DELIM после деления будет содержать частное и остаток. Команда для деления будет следующей:

Название	Операция	Операнды
	DP	DELIM, DELIT

Содержимое области DELIM после деления станет 00136C030C. Это значит, что при делении 4246 на 31 получается частное 136 и остаток 30. Делитель имеет длину два байта, поэтому остаток занимает два самых правых байта. Частное записывается в оставшееся место области для результата. При выполнении деления для решения вопроса о длинах можно руководствоваться следующим правилом: число байт в делимом равно числу байт в делителе плюс число байт в частном. Если частное не вмещается в отведенное для него поле, то произойдет прерывание.

Сдвиг десятичных данных. Команды сдвига не предусмотрены в наборе команд десятичной арифметики. Сдвиг десятичных чисел реализуется с помощью команд пересылки.

Рассмотрим сначала выполнение простейшего сдвига: сдвига вправо десятичного упакованного числа на четное число десятичных цифр. Предположим, что имеется упакованное десятичное число (например, 123456789C), состоящее из девяти цифр и занимающее область ISHOD длиной 5 байт. Необходимо переслать это число в область SDVIN длиной 5 байт, в которой находится число X'9999999999', предварительно опустив из пересылаемого числа две последние цифры и записав два нуля слева, т. е. сдвинуть на две цифры вправо. Сдвиг можно сделать двумя способами: изменяя и не изменяя первоначальное содержимое области ISHOD. Если не изменять содержимое области ISHOD, то сдвиг можно выполнить с помощью следующих команд (в поле

комментариев показано содержимое области SDVIN после выполнения каждой команды):

Название	Операции	Операнды
	MVC	SDVIN+1(4),ISHOD 9912345678
	MVN	SDVIN+4(1),ISHOD+4 991234567C
	MVC	SDVIN(1),ZERO ØØ1234567C
ZERO	DC	X'Ø'
ISHOD	DC	P'123456789'
SDVIN	DC	X'9999999999'

Первая команда MVC пересылает первые 4 байта области ISHOD в область SDVIN (начиная со второго байта области SDVIN). При этом в область SDVIN не пересылается последняя десятичная цифра и знак числа, первый байт области SDVIN не изменяется. Следующая команда MVN пересылает в область SDVIN знак числа. Последняя команда MVC пересылает один байт константы, имя которой ZERO, в первый байт области SDVIN. Константа ZERO содержит нули, поэтому в первом байте области SDVIN устанавливаются нули.

Если первоначальное содержимое области ISHOD в дальнейшем не используется, то можно выполнить поставленную задачу с помощью только двух следующих команд:

Название	Операция	Операнды
	MVN	ISHOD+3(1),ISHOD+4 1234567C9C 9999999999
ZAP		SDVIN,ISHOD(4) 1234567C9C ØØ1234567C

В комментариях указано содержимое областей ISHOD и SDVIN соответственно, после выполнения каждой команды. Команда MVN пересылает знак в байт, в котором он должен содержаться после сдвига. Команда ZAP выбирает 4 байта из области ISHOD и прибавляет их к содержимому SDVIN, предварительно очистив область SDVIN. В команде ZAP должна быть указана длина каждого операнда. Для первого операнда используется неявная длина, равная 5, для второго операнда указана явная длина, равная 4, чтобы не выполнялась пересылка двух последних цифр из области ISHOD.

Рассмотрим теперь более сложный пример, когда необходимо выполнить сдвиг вправо на нечетное число цифр. Здесь нужно применить специальную десятичную команду, предназначенную для этой цели, — команду MVO (ПЕРЕСЫЛКА СО СДВИГОМ). Действие этой команды заключается в следующем: знак (четыре младшие разряда) первого операнда не изменяется, второй операнд помещается слева от четырех младших разрядов первого опе-

ранда вплотную к этим разрядам. Таким образом, получается, что младшие четыре разряда первого операнда присваиваются в качестве младших разрядов ко второму операнду, все разряды второго операнда сдвигаются на 4 позиции влево, и полученный результат помещается на место первого операнда. Байты первого и второго операндов на правильность кодов не проверяются. В команде указывается длина каждого операнда. Если длина второго операнда меньше длины первого операнда, то он дополняется нулями от старших десятичных цифр. Если длина второго операнда больше длины первого операнда, то значащие цифры второго операнда, не помещающиеся в область первого операнда, игнорируются. Операнды могут перекрывать друг друга.

Рассмотрим числа, приведенные в предыдущем примере, но предположим, что необходимо выполнить сдвиг вправо на три десятичных цифры вместо двух. Это можно сделать с помощью следующих команд (в комментариях здесь и в дальнейших примерах показано только содержимое области SDVIN, так как область ISHOD не изменяется):

Название	Операция	Операнды	
	MVO	SDVIN,ISHOD(3)	0001234569
	MVN	SDVIN+4(1),ISHOD+4	000123456C

В команде MVO используется неявная длина первого операнда, равная 5, и явная длина второго операнда, равная 3. По этой команде 3 байта из области ISHOD (123456) помещаются в область SDVIN слева от четырех самых правых разрядов этой области. Так как пересылается меньшее число байт, чем длина области SDVIN, то слева к пересылаемым цифрам добавляются недостающие нули. Команда MVN присваивает числу SDVIN знак числа ISHOD. Если результат сдвига требуется оставить в области ISHOD, то можно обойтись только одной командой MVO ISHOD, ISHOD(3).

Рассмотрим пример сдвига десятичного числа влево. Допустим, в области ISHOD длиной 3 байта находится число 12345C. Необходимо получить в области SDVIN это число с четырьмя нулями справа. Возможная последовательность команд показана ниже:

Название	Операция	Операнды	
	MVC	SDVIN(3),ISHOD	12345C9999
	MVC	SDVIN+3(2),ZEROS	12345C0000
	MVN	SDVIN+4(1),SDVIN+2	12345C000C
	MVN	SDVIN+2(1),ZEROS	123450000C
SDVIN	DC	X'999999999'	
ISHOD	DC	P'12345'	
ZEROS	DC	X'0000'	

В первой команде MVC указана явная длина, так как нужно переслать 3 байта, а при использовании неявной длины имени SDVIN будут пересылаться 5 байт. При выполнении этой команды последние два байта области SDVIN не меняются, вторая команда MVC очищает их. Первая команда MVN помещает знак исходного числа в последний байт области SDVIN. Вторая команда MVN пересылает нули в те разряды числа, в которые попал лишний знак после первой пересылки исходного числа.

Рассмотрим сдвиг влево на нечетное количество десятичных цифр. Предположим, что сохраняются все условия предыдущего примера, только исходное число нужно сдвинуть влево не на четыре десятичных разряда, а на три. Это можно выполнить следующими командами:

Название	Операция	Операнды	
	MVC	SDVIN(3), ISHOD	12345C9999
	MVC	SDVIN+3(2), ZEROS	12345C0000
	MVN	SDVIN+4(1), SDVIN+2	12345C0000C
	NI	SDVIN+2, X'F0'	1234500000C
	MVO	SDVIN(4), SDVIN(3)	0123450000C

Первые три команды те же, что и в предыдущем примере. Для устранения лишнего знака использована команда NI вместо MVN. Команда NI имеет формат SI. Здесь операция выполняется для операндов, один из которых непосредственно записан в команде, а другой находится в основной памяти. Эта команда выполняет поразрядное логическое умножение. При выполнении этой операции разряд результата принимает значение 1, если соответствующие разряды каждого из операндов равны 1. Во всех остальных случаях разряд результата устанавливается в 0. В данном примере один операнд, представляющий собой непосредственный операнд, равен 11110000, другой операнд, адрес которого SDVIN+2, равен 01011100. Необходимо оставить без изменения первые 4 разряда байта SDVIN+2, поэтому первые четыре разряда непосредственного операнда — единицы, а остальные — нули. Разряды, равные нулю в непосредственном операнде, установят нули в последних четырех разрядах байта SDVIN+2. Последняя команда выполняет сдвиг на одну цифру вправо.

Преобразование формата десятичных данных. Предположим, в памяти находится число 124 в десятичном формате с зоной (ZON) и число 52, представленное как число с фиксированной точкой (DVOI). Необходимо получить сумму этих чисел в десятичном формате с зоной в области SUM. Поставленную задачу можно выполнить с помощью следующей программы, использующей команды РАСК (УПАКОВАТЬ) и UNPK (РАСПАКОВАТЬ), которые выполняют преобразование формата десятичных данных:

Название	Операция	Операнды	Название	Операция	Операнды
BEGIN	BALR	15, Ø		SVC	14
	USING	*15		DS	ØF
	PACK	RAB, ZON	ZON	DC	Z'124'
	L	6, DVOI	RAB	DS	PL2
	CVD	6, DEC	DVOI	DC	F'52'
	AP	DEC, RAB	DEC	DS	D
	UNPK	SUM, DEC	SUM	DS	ZL3
				END	BEGIN

Команда **PACK** преобразует число 124 из десятичного формата с зоной в упакованный формат. Второй адрес в этой команде указывает адрес числа в формате с зоной, первый адрес указывает адрес области, куда необходимо поместить число в упакованном формате. Команда **PACK** не принимает во внимание все зоны байт числа, за исключением зоны перед самой младшей цифрой, которая рассматривается как знак. Знак помещается в правые четыре разряда младшего байта области результата, а в остальной части этой области вплотную друг к другу размещаются цифры. Число при пересылке не изменяется, но оно может потерять левые значащие цифры, если они не вмещаются в область первого операнда, или дополняться слева нулями, если его длина меньше первого операнда. Поля операндов могут перекрываться. В результате выполнения команды **PACK** в приведенной программе десятичное число 124 в формате с зоной (F1F2C4) будет размещено в области RAB в упакованном формате (I24C).

Команда **CVD** преобразует двоичное число (число с фиксированной точкой), которое предварительно помещается в регистр 6, в десятичное число в упакованном формате и помещает его в область DEC.

После того как оба числа приведены к упакованному десятичному формату, можно выполнять команду **AP** (СЛОЖЕНИЕ ДЕСЯТИЧНОЕ), которая оперирует с числами именно такого формата. Результат сложения будет представлять собой десятичное число в упакованном формате, размещенное в области DEC. Но по условию требовалось получить результат в десятичном формате с зоной.

Команда **UNPK** выполняет преобразование числа из упакованного формата в десятичный формат с зоной. Цифры и знак упакованного операнда не меняются. Ко всем цифрам добавляется зона 1111. Исключение составляет самая младшая цифра, которая помещается в один байт со знаком. Правильность кодов знака и цифр операнда не проверяется. Второй операнд, если необходимо, перед преобразованием слева дополняется нулями. Значащие цифры второго операнда, не помещающиеся в поле первого операнда, игнорируются. Поля первого и второго операндов могут перекрываться. В результате выполнения команды **UNPK**, записанной в

программе, десятичное число из области DEC будет преобразовано из упакованного десятичного формата в десятичный формат с зоной и помещено в область SUM.

3.4.4. Команды с плавающей точкой

При разборе команд, выполняющих операции над числами с фиксированной точкой и десятичными числами, за правильной установкой точки (за определением целой части чисел) должен был следить сам программист. В научных и технических расчетах необходимо предоставить эту функцию ЭВМ. Если использовать при таких расчетах только команды с фиксированной точкой или десятичные команды, то программист должен все время следить за количеством цифр в величинах, возникающих в процессе вычисления. Необходимо знать максимальные возможные размеры всех данных, промежуточных результатов, а часто и их минимальные размеры. Все это нужно для того, чтобы избежать превышения емкости регистров и областей памяти. Получить все сведения о промежуточных вычислениях в программе часто бывает трудно, а иногда и невозможно. Работа с фиксированной точкой трудоемка и из-за того, что требуется выравнивать десятичные и двоичные числа с помощью сложных процессов сдвига.

По этим причинам большим удобством является предоставленная в ЕС ЭВМ, как и во многих других машинах, возможность выполнения операций над числами с плавающей точкой с помощью команд с плавающей точкой. Во время выполнения этих команд за положением точки следит сама ЭВМ. При использовании команд с плавающей точкой экономится время программирования, появляется возможность решения задач, которые нельзя решить с помощью команд с фиксированной точкой.

Числа с плавающей точкой представляются с помощью двух элементов: дроби (мантиссы) и порядка. Порядок — это степень числа 16, на которую нужно умножить дробь, чтобы получить представляемое число. В ЕС ЭВМ вместо порядка используется характеристика, которая численно равна порядку, увеличенному на 64 (характеристика всегда получается положительной, и в данных не нужно отводить разряд для изображения знака). Дробная часть числа, которая умножается на число 16 в степени, равной порядку, называется мантиссой. ЕС ЭВМ допускает два вида чисел с плавающей точкой, называемых соответственно короткими и длинными. В каждом из этих видов чисел в первом байте содержится характеристика. В коротком числе с плавающей точкой мантисса состоит из шести шестнадцатеричных цифр, которые содержатся в трех байтах, следующих за характеристикой. В длинном числе с плавающей точкой дробная часть состоит из 14 шестнадцатеричных цифр, содержащихся в семи байтах. Следовательно, короткое число занимает слово, а длинное — двойное слово. В языке Ассемблера предусмотрена возможность определять как

короткие, так и длинные числа с плавающей точкой с помощью команды Ассемблера DC (см. 3.6).

К средствам обработки чисел с плавающей точкой в ЕС ЭВМ относятся команды с плавающей точкой и регистры с плавающей точкой. В набор команд с плавающей точкой входят команды загрузки, сложения, вычитания, сравнения, умножения, деления и записи в основную память коротких и длинных чисел. Операции над короткими числами обычно выполняются быстрее и требуют меньшего объема памяти для хранения данных, чем операции над длинными числами. Использование длинных чисел позволяет получить повышенную точность вычислений.

Команды с плавающей точкой могут быть формата RR и RX. Первый операнд в команде — это номер регистра с плавающей точкой, над содержимым которого будет выполняться операция. Второй операнд в команде формата RR — это тоже номер регистра с плавающей точкой, а в команде формата RX — адрес памяти. Номера регистров, которые указываются в командах с плавающей точкой, могут быть только 0, 2, 4 и 6. Регистры с плавающей точкой имеют длину 64 разряда (двойное слово). Адрес памяти, указываемый в команде, должен быть расположен на границе слова, если число короткое, или на границе двойного слова, если число длинное. Транслятор проверяет, правильно ли записаны в командах с плавающей точкой номера регистров и находятся ли числа в памяти на границе слова или двойного слова.

Для получения максимальной точности арифметические операции над числами с плавающей точкой (сложение, вычитание, умножение и деление) выполняются с нормализацией результата. Нормализованное число с плавающей точкой имеет отличную от нуля старшую шестнадцатеричную цифру мантииссы. Число не нормализовано, если одна или более старших цифр мантииссы равны нулю. Процесс нормализации заключается в сдвиге мантииссы влево на одну шестнадцатеричную позицию за один раз до тех пор, пока старшая шестнадцатеричная цифра мантииссы не будет отлична от нуля, и в уменьшении характеристики на величину, равную числу сдвигов. Поскольку нормализация производится для шестнадцатеричных цифр, три старших двоичных разряда нормализованного числа могут быть нулями. Операции с плавающей точкой могут выполняться как с нормализацией результата, так и без его нормализации. Если складываются равные числа с разными знаками, то результат операции будет нулем. Это называется полной потерей значимости и при необходимости вызывает прерывание программы. Знак суммы определяется по алгебраическим правилам. Знак суммы с нулевой мантииссой всегда положителен. При выполнении большинства операций с плавающей точкой устанавливается признак результата. Для большинства операций значения признака результата 0, 1 или 2 указывают на то, что результат соответственно равен нулю, меньше нуля или больше нуля. Нулевым считается результат, имеющий нулевую ман-

тиссу. Значение признака результата 3 устанавливается в том случае, если произошло переполнение порядка.

При выполнении команд с плавающей точкой возможны прерывания, которые могут вызвать следующие причины: защита памяти, адресация, спецификация, переполнение порядка, исчезновение порядка, потеря значимости, деление с плавающей точкой.

Прерывание по защите памяти возникает, если ключ памяти операнда не совпадает с ключом защиты в слове состояния программы.

Если команда обращается к байту, отсутствующему в данной ЭВМ, происходит прерывание по адресации.

Прерывание по спецификации происходит, если в качестве регистра с плавающей точкой указан регистр с номером, отличным от номера 0, 2, 4 и 6, или если операнд длиной в слово (двойное слово) не расположен в памяти на границе слова (двойного слова).

Прерывание по переполнению порядка возникает, если при выполнении операции происходит переполнение порядка (характеристика результата превышает 127), а мантисса результата при этом отлична от нуля.

Если при выполнении арифметической операции с плавающей точкой происходит исчезновение порядка (характеристика результата оказалась меньше нуля), а мантисса результата при этом отлична от нуля, то возникает прерывание по исчезновению порядка. Это прерывание можно замаскировать в слове состояния программы.

Прерывание по потере значимости возникает, если при сложении или вычитании с плавающей точкой мантисса результата равна нулю. Это прерывание можно замаскировать в слове состояния программы.

Если мантисса делителя равна нулю, происходит прерывание по делению с плавающей точкой.

В наборе машинных команд существуют 44 команды с плавающей точкой, но функций, которые они выполняют, гораздо меньше. Например, существуют 8 команд, выполняющих сложение; 8 команд, выполняющих вычитание. Необходимую команду выбирает программист исходя из имеющихся у него данных (длинных или коротких), местонахождения данных (оба в регистрах или в регистре и в памяти), требующейся точности (выполнение действий с нормализацией или без нормализации).

Рассмотрим пример. Допустим, имеется следующая формула:

$$X = \left(\frac{A + \frac{B-C}{2}}{4,25 - 2K} \right)^2$$

Необходимо получить значение X при заданных значениях параметров: A=5,9; B=4,1176; C=3,279; K=0,6904. Программа вычисления этой функции может быть записана следующим образом:

Название	Операция	Операнды
BEGIN	BALR	15,0
	USING	*,15
	LE	2,K вычисление
	ME	2,DB знаменателя
	LCER	2,2
	AE	2,KONST
	LE	4,B вычисление
	SE	4,C числителя
	HER	4,4
	AE	4,A
	DER	4,2 деление числителя на знаменатель
	MER	4,4 возведение в степень
	STE	4,X
	SVC	14
A	DC	E'5,9'
B	DC	E'4.M76'
C	DC	E'3.279'
K	DC	E'6904E-4'
DB	DC	E'2'
KONST	DC	E'4.25'
X	DS	E
	END	BEGIN

Первая команда, с которой начинаются вычисления, — это команда LE (ЗАГРУЗКА КОРОТКАЯ). По команде LE число с плавающей точкой с именем K (0,6904) помещается в регистр с плавающей точкой 2. Эта команда обращается к регистру 2 с плавающей точкой, а не к общему регистру 2, потому что код операции LE указывает команду с плавающей точкой, которая может использовать только регистр с плавающей точкой. Регистры с плавающей точкой имеют длину 64 разряда. По команде LE, которая загружает короткий операнд, число с именем K будет загружено в левую половину регистра (32 разряда), правая половина регистра будет оставаться без изменения. Если бы число было определено как длинное число с плавающей точкой (оператором DC с типом D), то нужно было бы использовать команду LD, по которой загружались бы все 64 разряда регистра длинным числом с плавающей точкой. Для загрузки числа с плавающей точкой в регистр с плавающей точкой не из памяти, а из другого регистра с плавающей точкой можно использовать команды LER или LDR формата RR (соответственно для короткого или длинного числа).

Следующая команда — команда ME (УМНОЖЕНИЕ). По этой команде содержимое регистра 2 с плавающей точкой, в котором находится число 0,6904, умножается на 2. Результат помещается в регистр 2. В противоположность умножению с фиксированной точкой для выполнения умножения с плавающей точкой требуется только один регистр. Умножение длинных чисел с плавающей точкой можно было бы выполнить командой MD. Если бы оба множителя были расположены в регистрах с плавающей точкой,

кой, то можно было бы воспользоваться командами MER или MDR формата RR.

Команда LCER (ЗАГРУЗКА ДОПОЛНЕНИЯ) формата RR изменяет значение знакового разряда второго операнда на противоположный, характеристика и мантисса не изменяются. Результат операции помещается на место первого операнда. В данном примере эта команда изменяет знак произведения. Произведение остается в регистре 2, хотя эту команду можно использовать и для регистров с разными номерами.

Команда AE (СЛОЖЕНИЕ С НОРМАЛИЗАЦИЕЙ КОРОТКОЕ) формата RX прибавляет к произведению число 4,25. В данном примере вычисляется сумма коротких чисел с плавающей точкой. Для сложения длинных чисел с плавающей точкой можно было бы использовать команду AD, для сложения двух чисел, находящихся в регистрах с плавающей точкой, — команду AER или ADR. Результат сложения сохраняется в регистре 2 с плавающей точкой.

Далее подсчитывается числитель. Загрузка числа B (4,1176) в регистр 4 с плавающей точкой и вычитание числа C (3,279) выполняются с помощью команд LE и SE. В программе деление на 2 выполнено не обычной командой деления, а командой HER (ДЕЛЕНИЕ ПОПОЛАМ). Команда HER используется для коротких чисел с плавающей точкой (для длинных используется команда HDR). Это команды формата RR. По команде HER (или HDR) второй операнд делится на 2 и результат помещается на место первого операнда. Команда HER, записанная в программе, вызывает деление числа в регистре 4 на 2 и размещение результата в регистре 4. Результат не нормализуется. Как это часто бывает в командах формата RR, здесь в качестве первого и второго операндов используется один и тот же регистр.

Следующая команда AE выполняет сложение числа в регистре 4 с числом A (5,9). Результат этой операции будет нормализованным числом. После выполнения рассмотренных команд в регистре 4 с плавающей точкой будет находиться значение числителя, в регистре 2 — значение знаменателя. Далее выполняется деление чисел с плавающей точкой с помощью команды DER формата RR (ДЕЛЕНИЕ КОРОТКОЕ). По этой команде делимое, находящееся в регистре 4, делится на число, находящееся в регистре 2, и частное помещается на место делимого в регистр 4. Если делитель находится в памяти, можно использовать для деления коротких чисел команду DE формата RX. Для деления длинных чисел с плавающей точкой можно использовать команду DDR формата RR, если делитель находится в регистре, или команду DD формата RX, если делитель находится в памяти. При делении коротких чисел содержимое младших 32 разрядов регистра с плавающей точкой игнорируется и остается без изменения.

В заключение в программе выполняется возведение в степень результата деления, который содержится в регистре 4. Это выполняется командой MER (УМНОЖЕНИЕ) формата RR, в которой

оба операнда содержатся в регистре 4. И, наконец, команда STE (ЗАПИСАТЬ В ПАМЯТЬ) помещает результат в основную память в область X. Длинное число с плавающей точкой можно сохранить в основной памяти с помощью команды STD.

В приведенном примере были рассмотрены в основном все команды с плавающей точкой. Некоторые другие команды, например команды сравнения, будут рассмотрены при разборе команд переходов.

3.4.5. Логические команды

Набор команд ЕС ЭВМ предусматривает команды для логической обработки данных. Операнды таких команд рассматриваются, как группы восьмизрядных байт. Операнды могут находиться в памяти или в общих регистрах, некоторые операнды присутствуют в самой команде. Они могут быть длиной в слово, двойное слово или иметь переменную длину от 0 до 256 байт.

Логическими командами выполняются поразрядные операции, проверка разрядов, пересылка, сравнение, перекодировка, редактирование и сдвиг.

Логические команды могут быть любого формата: RR, RX, RS, SI или SS. Команды формата RR, RX и RS оперируют с данными длиной в слово. В командах формата SS используется длина, одинаковая для обоих операндов (длина первого операнда, которая может достигать 256 байт). Команды формата SI выполняют операции над однобайтовыми операндами. Для большинства команд формата RX, выполняющих логические операции, данные должны быть расположены на границе слова. В результате выполнения всех операций логического сравнения, поразрядных операций, а также операций проверки и редактирования устанавливается признак результата. Для операций логического сравнения значения признака результата 0, 1 или 2 указывают, что первый операнд соответственно равен, меньше или больше второго. Для поразрядных операций значение 0 или 1 указывает соответственно на нулевой или не нулевой результат. Для команд редактирования, перекодировки и проверки устанавливается признак результата, характерный для каждой команды. Об установке признака результата для таких команд будет сказано при их непосредственном разборе.

Прерывание при выполнении логических команд вызывают следующие причины: защита памяти, адресация, спецификация и данные.

Прерывание по защите возникает, если ключ памяти операндов не совпадает с ключом защиты в слове состояния программы.

Если команда обращается к байту памяти, отсутствующему в данной ЭВМ, возникает прерывание по адресации.

Прерывание по спецификации возникает, если для пары общих регистров, содержащих 64-разрядный операнд, указан номер не-

четного регистра или, если операнд длиной в слово, в памяти начинается не с границы слова.

Если в десятичном данном, используемом командой, имеется неправильный код цифры, возникает прерывание по данным.

Логические команды (CLR, CL, CLI, CLC, TM, TS), выполняющие сравнение и логические проверки, будут рассмотрены подробнее вместе с командами перехода, что даст более наглядное представление об их применении. При логической обработке данных могут использоваться также некоторые команды с фиксированной точкой, например, для записи данного в память или загрузки данных в регистр.

Команды пересылки. Команды пересылки выполняют перемещение данных в основной памяти. Допустим, в памяти имеются 2 байта информации. На основании этих байт необходимо построить запись длиной 8 байт по следующим правилам: в первый байт поместить значение C'A' (признак начала записи); во второй и третий байты поместить исходные байты; в старшие 4 разряда четвертого и пятого байт поместить нули, а в младшие — соответствующие младшие разряды исходных байт; в старшие 4 разряда шестого и седьмого байт поместить соответствующие старшие разряды исходных байт, а в младшие — нули; в последний байт поместить значение X'FF' (признак конца записи). Эту задачу можно выполнить с помощью следующей программы, использующей команды пересылки:

Название	Операция	Операнды
BEGIN	BALR	15,0
	USING	*,15
	MVI	NOV,C'A'
	MVC	NOV+1(2),ICX
	MVI	NOV+3,X'00'
	MVC	NOV+4(3),NOV+3
	MVN	NOV+3(2),ICX
	MVZ	NOV+5(2),ICX
	MVI	NOV+7,X'FF'
	SVC	14
ICX	DC	C'A1B2'
NOV	DC	XL8'00'
	END	BEGIN

В приведенной программе исходная информация определяется оператором DC с именем ICX, область длиной 8 байт для результата определяется оператором DC с именем NOV.

Для построения необходимой записи в программе используется несколько команд MVI формата SI, в которых байт для пересылки указывается в самой команде. С помощью команд MVI значение C'A' помещается в первый байт области NOV, значение X'00' — в четвертый байт; значение X'FF' — в последний.

Первая команда MVC, используемая в программе, выполняет пересылку двух исходных байт во второй и третий байт области NOV. В этой команде указана явная длина. Если бы понадобилось пересылать больше 256 байт, то одной команды было бы недостаточно, так как в команде MVC можно указать длину не более 256. Вторая команда MVC используется для пересылки нулей в пятый, шестой и седьмой байты области (указана явная длина, равная 3). На этой команде можно увидеть, что один символ можно записать по всей области, указав начало области первого операнда на один байт правее начала области второго операнда, потому что команда MVC выполняет пересылку не сразу всего поля, а по одному байту.

Команды MVN (ПЕРЕСЫЛКА ЦИФР) и MVZ (ПЕРЕСЫЛКА ЗОН) пересылают только часть байт; в первом случае младшие 4 разряда (цифра) каждого байта второго операнда помещаются в младшие разряды соответствующих байт первого операнда, во втором — 4 старших разряда (зона) помещаются в старшие разряды. В первом случае старшие разряды, а во втором — младшие остаются неизменными в обоих операндах. Таким образом, в четвертый и пятый байты области NOV командой MVN будут помещены младшие разряды исходных байт, а в пятый и шестой командой MVZ — старшие разряды этих байт. В результате в области NOV будет следующая информация X'C1A1B20102A0B0FF'.

Действия над битами. Для работы с отдельными битами предназначены логические команды: И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ. Эти команды позволяют установить значение некоторого разряда в байте, равным 0 или 1, или заменить значение разряда на противоположное.

Команды ИЛИ (OR, O, OI, OC) выполняют поразрядное логическое сложение операндов, результат помещается на место первого операнда. В зависимости от места нахождения операндов применяется необходимая команда: OR, если оба операнда находятся в регистре; O, если один операнд — в регистре, а другой — в памяти; OI, если один операнд находится непосредственно в команде, а другой — в памяти, OC, если оба операнда находятся в памяти. В результате операции логического сложения в тех битах, в которых хотя бы один из операндов имел 1, установится 1.

Команды И (NR, N, NI, NC) выполняют поразрядное логическое умножение операндов, результат помещается на место первого операнда. Аналогично команде ИЛИ в зависимости от местонахождения операндов применяется та или другая команда. Команда И записывает единицы только в те разряды результата, которые в обоих операндах равны единице, а остальные разряды устанавливаются в нуль.

Команды ИСКЛЮЧАЮЩЕЕ ИЛИ (XR, X, XI, XC) выполняют поразрядное сложение по модулю 2 (поразрядное сложение без переноса). Результат помещается на место первого операнда. В зависимости от местонахождения операндов применяется та или

другая команда. В результате операции в тех битах, где у обоих операндов были различные значения (в одном 0, а в другом 1), и наоборот, будут единицы, а в тех битах, где у обоих операндов были одинаковые значения (либо 0, либо 1), будут нули.

По командам И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ операции выполняются над каждой парой соответствующих разрядов двух операндов. Результаты, получаемые после выполнения операций в зависимости от значения разрядов, приведены в табл. 1.

Таблица 1

Операция	Первый операнд	Второй операнд	Результат
И	1	0	0
	0	1	0
	0	0	0
	1	1	1
ИЛИ	1	0	1
	0	1	1
	0	0	0
	1	1	1
ИСКЛЮЧАЮЩЕЕ ИЛИ	1	0	1
	0	1	1
	0	0	0
	1	1	0

Команда И может быть использована для того, чтобы установить некоторый бит в нуль. Допустим, в байте SIM необходимо биты 1, 3, 6, 7 установить в 0 (биты байта пронумерованы слева направо от 0 до 7). Для этого необходимо подготовить двоичную константу 10101100, т. е. константу с нулями в тех разрядах, где необходимы нули, и с единицами в тех разрядах, где необходимо оставить прежнее значение. Если нужно выполнить операцию только с одним байтом, можно использовать команду SI. Если нужно записать константу в шестнадцатеричном виде, команда, устанавливающая нули в битах 1, 3, 6, 7 байта SIM, будет выглядеть следующим образом: NI SIM, X'AC'.

Если требуется установить некоторый бит в единицу, можно использовать команду ИЛИ. Например, необходимо те же разряды в байте SIM установить в единицу. Тогда следует подготовить двоичную константу 01010011, которая содержит единицы в тех разрядах, где необходимо установить единицы, и нули, где нужно оставить прежнее значение, и выполнить следующую команду: OI SIM, X'53'.

Чтобы инвертировать значение разряда, можно использовать команду ИСКЛЮЧАЮЩЕЕ ИЛИ. Для этого нужно написать та-

кую константу, где были бы единицы в тех битах, которые необходимо инвертировать, и нули в битах, которые должны остаться неизменными, а затем выполнить команду ИСКЛЮЧАЮЩЕЕ ИЛИ. Например, если в байте SIM необходимо инвертировать разряды 1, 3, 6, 7, можно записать следующую команду: XI SIM, X'53'.

Действия над символами. При обработке логической информации есть возможность выполнять действия над отдельными символами. Например, допустим, в памяти имеется байт, в котором содержится шестнадцатеричное число X'78'. В этом числе необходимо шестнадцатеричную цифру 8 заменить цифрой С (шестнадцатеричное представление знака плюс для десятичного числа), и полученное число поместить в другой байт памяти. Такое действие можно выполнить с помощью следующей программы:

Название	Операция	Операнды
BEGIN	BALR	15,0
	USING	*16
	IC	5,SHEC
	N	5,—X'FFFFFF0'
	O	5,—X'0000000C'
	STC	5,RES
	SVC	14
SHEC	DC	X'78'
RES	DS	X
	END	BEGIN

Вычисления в программе начинаются с команды IC (ПРОЧИТАТЬ СИМВОЛ). Это команда формата RX. Она извлекает один символ (байт) из памяти и помещает его в самые младшие 8 разрядов регистра. Содержимое остальных разрядов регистра при выполнении этой команды не изменяется. В нашем случае байт X'78' с адресом SHEC помещается в последние 8 разрядов регистра 5. Содержимое остальных разрядов регистра 5 неизвестно.

Команда N (И) устанавливает в нуль последние 4 бита регистра 5, так как в константе, являющейся вторым операндом и определяемой как литерал, последние четыре бита равны нулю, а остальные — 1. Значения разрядов регистра, которым соответствуют единицы в константе, не изменяются. В результате выполнения команды N младшие 8 разрядов регистра 5 будут содержать значение X'70', содержимое остальных разрядов этого регистра не изменится.

Команда O устанавливает в последних четырех битах регистра 5 значение B'1100', потому что вторым операндом этой команды является константа X'0000000C'. При выполнении команды те биты первого операнда, которые соответствуют нулевым битам второго, остаются без изменения, а те биты первого операнда, которые соответствуют единичным битам второго, устанавливаются в едини-

цу. Таким образом, после выполнения команды О младшие разряды регистра 5 будут содержать значение $X'7C'$. По условию это значение необходимо сохранить в памяти. Запись одного символа (байта) в память выполняет команда STC.

Команда STC имеет формат RX. Эта команда содержит содержимое разрядов 24—31 регистра помещает в память. Содержимое остальных разрядов регистра при этом безразлично. В приведенной программе по команде STC в память в байт с адресом RES помещаются последние 8 разрядов регистра 5. В этих разрядах находится как раз значение $X'7C'$. Таким образом, в байт с адресом RES будет помещена необходимая константа.

Загрузка адреса. В состав логических команд входит команда LA (ЗАГРУЗКА АДРЕСА). По команде LA адрес второго операнда помещается в младшие 24 разряда (разряды 8—31) общего регистра, определяемого первым операндом. Загружается в регистр именно адрес второго операнда, а не содержимое области памяти с этим адресом. В остальные разряды общего регистра (разряды 0—7) помещаются нули. Хотя команда LA имеет формат RX, обращение к памяти по адресу, указанному в команде, отсутствует. Вычисление значения, загружаемого в регистр, выполняется по правилам вычисления адреса исходя из содержимого регистра базы, регистра индекса и смещения.

Общий регистр, указываемый в качестве регистра базы, регистра индекса или первого операнда в команде LA, может быть одним и тем же. Поэтому команда LA часто используется в случаях, когда необходимо увеличить содержимое общего регистра на некоторое число, не превышающее 4095. Например, для увеличения содержимого общего регистра 5 на 10 можно использовать следующую команду: LA 5,10(5). В этой команде используется явный неиндексируемый адрес. Адрес, который команда загружает в регистр 5, вычисляется как содержимое регистра базы 5 плюс смещение, равное 10. Таким образом, содержимое регистра 5 увеличивается на 10 и загружается в регистр 5. Такой же результат будет получен при выполнении команды LA 5,10(5,0), в которой используется явный индексируемый адрес с регистром базы 0 и регистром индекса 5.

С помощью команды LA можно также выполнять сложение чисел с фиксированной точкой, например, к числу в регистре добавить число, не превышающее 4095, и результат поместить в другой регистр. Так, по команде LA 8,10(5) к содержимому регистра 5 будет добавлено число 10, а результат помещен в регистр 8.

Кроме того, с помощью команды LA можно выполнить сложение чисел, находящихся в регистрах, к их сумме добавить число, не превышающее 4095, и полученный результат поместить в общий регистр. Например, по команде LA 8,712(3,4) к содержимому регистра 3 будет добавлено содержимое регистра 4, полученная сумма увеличится на 712 и окончательный результат будет помещен в регистр 8.

Командой LA можно выполнить загрузку в общий регистр це-

лого числа, не превосходящего 4095. Для загрузки числа 100 в регистр 8 можно использовать следующую команду: LA 8,100. Адресом в этой команде является абсолютное значение 100, которое меньше 4096. Поэтому регистром базы для этого адреса является регистр 0, а смещение равно 100. При выполнении команды в регистр 8 будет загружено число 100 (сумма смещения и содержимого регистра базы, равного 0 для нулевого регистра).

Команды сдвига. Рассмотрим пример, в котором используются команды сдвига. Допустим, в слове основной памяти содержатся три элемента: разряды 0—11 содержат элемент А, разряды 12—23 — элемент В, разряды 24—31 — элемент С. Требуется переслать каждый элемент в отдельную область памяти длиной в полуслово. Программа, выполняющая эту задачу, может выглядеть следующим образом:

Название	Операция	Операнды
BEGIN	BALR USING L	15,0 *,15 6,SLOVO 789ABCDE 00000000
PERV	SRDL	6,8 00789ABC DE000000
	SRL	7,24 00789ABC 000000DE
	STH	7,C 00789ABC 000000DE
	SRDL	6,12 00000789 ABC00000
SLED	SRL	7,20 00000789 00000ABC
	STH	7,B 00000789 00000ABC
	STH	6,A
	SVC	14
SLOVO	DC	X'789ABCDE'
A	DS	H
B	DS	H
C	DS	H
	END	BEGIN

В начале программы исходное слово X'789ABCDE' загружается в четный регистр 6, чтобы в дальнейшем можно было использовать команду двойного сдвига. В комментариях приведено содержимое регистров 6 и 7 после выполнения команды. Предполагается, что содержимое регистра 7 в начале выполнения программы равно нулю, хотя на выполнение программы оно никакого влияния не оказывает.

Команда SRDL (СДВИГ ВПРАВО ДВОИНОЙ КОДОВ) сдвигает первый операнд вправо на число разрядов, определяемое вторым операндом. Команда SRDL — команда формата RS. Первый операнд должен находиться в паре подряд стоящих общих регистров, первый из которых должен иметь четный номер. Номер четного регистра указывается в команде. Адрес второго операнда используется не как адрес памяти. Младшие шесть разрядов этого адреса указывают число разрядов, на которое нужно произвести

сдвиг; остальные разряды адреса игнорируются. В сдвиге участвуют все 64 разряда пары регистров, определяемых в команде. Младшие разряды, выдвигаемые за пределы нечетного регистра, теряются. Освобождающиеся старшие разряды четного регистра заполняются нулями.

Записанная в приведенной программе первая команда SRDL использует регистры 6 и 7. Содержимое этих регистров сдвигается вправо на 8 разрядов, при этом младшие 8 разрядов регистра 6 (X'DE') помещаются в начало регистра 7. Команда SRL (СДВИГ ВПРАВО КОДОВ) с именем PERV сдвигает содержимое регистра 7 на 24 разряда вправо и, таким образом, число X'DE' помещается в младшие разряды регистра 7. Это значит, что элемент С полностью выделен из исходного слова, и следующая команда STH (ЗАПИСЬ В ПАМЯТЬ ПОЛУСЛОВА) поместит его в отведенную область.

Команда SRL выполняется так же, как и команда SRDL, только в сдвиге участвуют лишь 32 разряда одного регистра 7. Команды SRDL и SRL являются командами логического сдвига. При выполнении этих команд сдвигаются все 64 или 32 разряда, и в освобождающиеся левые разряды записываются нули. При выполнении команд SRDA (СДВИГ ВПРАВО ДВОЙНОЙ) и SRA (СДВИГ ВПРАВО), которые относятся к командам с фиксированной точкой, знак первого операнда остается при сдвиге без изменения. Знак операнда находится в самом левом разряде регистра (или четного регистра), если выполнялась команда SRDA, а освобождающимся при сдвиге левым разрядам присваивается значение знакового разряда. Кроме того, после выполнения команд SRDA и SRA устанавливается признак результата. Команды арифметического сдвига SRDA и SRA обычно применяются для деления на число 2 в некоторой степени без сохранения остатка, потому что эти команды сдвига выполняются быстрее, чем команды деления.

Вторая команда SRDL, используемая в программе, сдвигает содержимое регистров 6 и 7 на 12 разрядов вправо. В младших разрядах регистра 6 будет находиться элемент А, а в старших разрядах регистра 7 — элемент В. Команда SRL с именем SLED сдвигает элемент В в младшие разряды регистра 7. Затем элементы В и А сохраняются в памяти.

Команда STH, выполняющая сохранение результатов в памяти, запоминает правые 16 разрядов регистра 7 (элемент В) в двух байтах области, адрес которой указан в команде. Это команда формата RX, но в нашем случае адрес, используемый в команде, не индексруется. Предположим, в условии поставленной ранее задачи добавлено следующее: значение самого левого разряда каждого элемента (0, 12 и 24 соответственно для элементов А, В и С) представляет собой знак элемента (0 — плюс, 1 — минус). При таком условии распространение знакового разряда на левые освобождающиеся при сдвиге разряды можно было бы выполнить с помощью команд арифметического сдвига. Команды, выполняющие сдвиг, выглядели бы следующим образом:

Название	Операция	Операнды
	L	6, SLOVO 789ABCDE ØØØØØØØØ
	SRDA	6,8 ØØ789ABC DEØØØØØØØØ
	SRA	7,24 ØØ789ABC FFFFFFFDE
	STH	7,C ØØ789ABC FFFFFFFDE
	SRDA	6,12 ØØØØØØ789 ABCFFFFFF
	SRA	7,2Ø ØØØØØØ789 FFFFFABC
	STH	7,B ØØØØØØ789 FFFFFABC
	STH	6,A ØØØØØØ789 FFFFFABC

Для сдвига обрабатываемых данных влево предназначены команды логического и арифметического сдвига влево — SLL, SLDL, SLA, SLDA. Эти команды аналогичны командам сдвига вправо.

Команды логического сдвига SLL (СДВИГ ВЛЕВО КОДОВ) и SLDL (СДВИГ ВЛЕВО ДВОЙНОЙ КОДОВ) выполняют сдвиг влево всех 32 (или 64 разрядов) общего регистра (или пары регистров). Старшие разряды, выдвигаемые за пределы регистра (или четного регистра), теряются, освобождающиеся младшие разряды заполняются нулями. Команды арифметического сдвига SLA (СДВИГ ВЛЕВО) и SLDA (СДВИГ ВЛЕВО ДВОЙНОЙ ВЛЕВО), относящиеся к командам с фиксированной точкой, выполняют сдвиг влево 31 или 63 соответственно разрядов. Знаковый разряд, т. е. самый левый разряд регистра (или четного регистра), при сдвиге не изменяется. В освободившиеся младшие разряды помещаются нули. Если при выполнении арифметического сдвига оказывается, что значение разряда, выдвигаемого на место знакового разряда, отличается от значения знакового разряда (знаковый разряд в этом случае должен измениться), то возникает переполнение. Команды SLA и SLDA вызывают установку значения признака результата.

Редактирование. В системе команд ЕС ЭВМ присутствуют команды, упрощающие подготовку данных для вывода. В частности, для этой цели предназначены команды редактирования: ED (ОТРЕДАКТИРОВАТЬ) и EDMK (ОТРЕДАКТИРОВАТЬ И ОТМЕТИТЬ).

По команде ED десятичное данное преобразуется из упакованного формата в формат с зоной. Кроме того, выполняется редактирование по шаблону. Шаблон — набор специальных символов, описывающих формат выводимых данных. Команда ED позволяет подавлять впереди стоящие нули, вставлять в число разделители (например, запятые, десятичные точки), упрощает заполнение полностью нулевых полей нужными знаками, позволяет отредактировать несколько чисел сразу, объединить числовую информацию с текстовой. Команда ED является командой формата SS, т. е. использует два адреса памяти. Второй адрес, указываемый в команде ED, должен быть адресом данного, которое нужно от-

редактировать. Первый адрес должен указывать адрес шаблона, который управляет редактированием.

Отредактированный результат после выполнения команды помещается на место шаблона. Допустимые в шаблоне символы и их влияние на выполнение команды рассматриваются ниже. Исходное данное представляет собой десятичное упакованное число и должно содержать правильные коды десятичных цифр и знака. При выполнении команды используется одна длина для двух операндов (длина первого операнда). Результат операции представляется в формате с зоной, поэтому длина первого операнда должна указываться с учетом длины результата в формате с зоной. Оба операнда обрабатываются слева направо, байт за байтом.

На то, какой символ помещается в область результата, влияют три фактора: цифра исходного данного; символ шаблона; состояние специального индикатора, называемого индикатором значимости.

В зависимости от этих факторов может быть выполнено одно из следующих действий:

- к цифре из исходного данного добавляется зона, и полученный символ заносится в область результата (заменяет символ шаблона);

- символ шаблона не меняется;

- в область результата заносится символ-заполнитель (первый символ шаблона).

Индикатор значимости устанавливается при выполнении команды и используется для того, чтобы управлять замещением символов шаблона (занесением результата в память). Он устанавливается в начале операции в нуль, а затем его состояние меняется в зависимости от исходной цифры и символа шаблона. Состояние индикатора значимости определяет символ, которым будет заменяться символ в шаблоне: цифрой или символом-заполнителем. Значение индикатора значимости, равное 0, обозначает, что в исходном данном не было найдено ни одной значащей (ненулевой) цифры, в этом случае символ шаблона замещается символом-заполнителем. Значение индикатора значимости, равное 1, обозначает, что либо в некотором предыдущем байте данных найдена значащая цифра, либо в шаблоне найден символ начала значимости (объяснение этого символа см. ниже). При таком значении индикатора значимости символ шаблона заменяется цифрой из исходного данного, даже если эта цифра — нуль.

Рассмотрим действие символа шаблона при выполнении команды редактирования. Имеются три символа шаблона со специальным назначением при редактировании: символ выбора цифры — X'20', символ начала значимости — X'21', символ разделения полей — X'22'. Эти символы записываются в шаблоне, а при выполнении команды редактирования замещаются либо цифрой из исходного данного, либо символом-заполнителем. Если символ шаблона является символом выбора цифры либо символом начала

значимости, то выбирается и анализируется очередная цифра из исходного данного.

Индикатор значимости, равный нулю, обозначает, что незначащие нули надо подавлять. В этом случае, если выбранная цифра — нуль, то символ шаблона заменяется символом-заполнителем, а не цифрой. Если же выбранная цифра — не нуль, то символ шаблона заменяется исходной цифрой, а индикатор значимости устанавливается в 1, указывая, что дальше все цифры будут значащими независимо от того, равны они нулю или нет. В случае присутствия в шаблоне символа начала значимости индикатор значимости всегда устанавливается в 1.

Таким образом, с помощью символа начала значимости в шаблоне программист может всегда указать, что нет необходимости подавлять незначащие нули. Символ разделения полей в шаблоне записывается, если редактируются несколько полей сразу. Этот символ заменяется символом-заполнителем, а индикатор значимости при встрече символа разделения полей устанавливается в нуль, в результате чего опять будет выполняться проверка на незначащие нули для последующих выбираемых исходных цифр. Все другие символы, записанные в шаблоне, кроме трех перечисленных выше, обрабатываются одинаково: если индикатор значимости равен единице, символ шаблона не меняется, если индикатор значимости равен нулю, символ шаблона заменяется символом-заполнителем.

Символ-заполнитель всегда является первым символом шаблона, используемого для данной команды ED. Символ-заполнитель остается без изменения в области результата, кроме случая, когда он совпадает с символом выбора цифры ($X'20'$) или с символом начала значимости ($X'21'$). В этом случае выполняется анализ исходной цифры, и если она не нуль, то она вставляется в область результата на место символа-заполнителя.

При выполнении команды выбирается последовательно каждый байт исходного данного. Левые четыре разряда байта анализируются первыми (проверяется, определяют ли они десятичную цифру), а правые хранятся до рассмотрения следующего символа шаблона. Если цифра помещается в область результата, к ней добавляется зона. Анализ, не находится ли в правых четырех разрядах код знака данного, производится сразу после анализа левых четырех разрядов. Если в правых четырех разрядах — код знака, то плюс вызывает установку индикатора значимости в нуль, а минус — в единицу, и в дальнейшем правые разряды больше не рассматриваются.

Для каждой очередной цифры исходного данного выбирается и анализируется очередной символ шаблона.

В результате выполнения команды ED устанавливаются следующие значения признака результата:

0 — при наличии только нулей в исходном данном вне зависимости от состояния индикатора значимости;

1 — при наличии ненулевых цифр в исходном данном и индикаторе значимости, равном 1 (исходное данное отрицательное и не равно 0);

2 — при наличии ненулевых цифр в исходном данном и индикаторе значимости, равном 0 (исходное данное положительное и не равно 0).

Операция редактирования может выполняться над несколькими полями, но установленный признак результата всегда относится только к полю, стоящему за последним символом разделения полей. Если последний символ шаблона является символом разделения полей, то признак результата устанавливается равным нулю. Результаты редактирования в зависимости от факторов, влияющих на редактирование, в общем виде представлены в табл. 2.

Таблица 2

Символ шаблона	Название символа шаблона	Анализ исходной цифры	Состояние индикатора значимости	Исходная цифра	Символ, помещаемый в результат	Переключение индикатора значимости
X'20'	Символ выбора цифры	Анализируется	0	Не 0	Цифра	1
				0	Символ-заполнитель	Не изменяется
			1	Любая	Цифра	Не изменяется
X'21'	Символ начала значимости	Анализируется	0	Не 0	Цифра	1
				0	Символ-заполнитель	1
			1	Любая	Цифра	Не изменяется
X'22'	Символ разделения полей	Не анализируется	Не анализируется	Любая	Символ-заполнитель	0
Другие символы	Включение текста	Не анализируется	0	Любая	Символ-заполнитель	Не изменяется
			1	Любая	Символ шаблона не изменяется	Не изменяется

Рассмотрим применение команды ED на примерах. Предположим, требуется подавить в числе старшие нули, при этом не нужно вставлять никаких знаков препиания. Редактируемое число находится в области DATA длиной 4 байта. Редактируемое число представляется в упакованном формате. Предположим, что число положительное. Знак данного при выполнении редактирования не рассматривается как цифра, а какое влияние он может оказать на редактирование, рассмотрим несколько позднее.

Предположим, что исходное число занимает 4 байта (7 цифр и знак числа), поэтому шаблон должен иметь длину 8 байт: 7 байт для числа и один байт для символа-заполнителя, который всегда должен быть самым первым в шаблоне. Символ-заполнитель, заменяющий первые незначащие нули, выбирается программистом (обычно это пробел). Символы шаблона, следующие за символом-заполнителем, нужно выбрать такими, чтобы нули числа, расположенные до первой значащей цифры, подавлялись, а остальные цифры числа помещались в шаблон. Операция редактирования будет выполняться необходимым образом, если в качестве этих символов шаблона использовать символ выбора цифры (X'20'). Если в области SHAB находится шаблон X'40202020202020', то редактирование числа из области DATE можно выполнить следующими командами:

Название	Операция	Операнды
	MVC ED	WORK,SHAB WORK,DATA

WORK — имя области памяти длиной 8 байт. После выполнения этих двух команд WORK будет содержать отредактированный результат. В области SHAB будет сохранен первоначальный шаблон, и его можно снова использовать для редактирования других данных.

В используемом шаблоне X'40' — код пробела, X'20' — код символа выбора цифры. Допустим, в DATA находится следующее число: X'0002000C' (C — код знака плюс). Результат редактирования будет иметь вид: X'404040F2F0F0F0', а на печати в символьном виде: $\circ\circ\circ\circ$ 2000, в нем все нули слева от первой ненулевой цифры заменены пробелами, а остальные цифры числа не изменились.

В шаблоне в качестве символа-заполнителя, находящегося в самом левом байте, может использоваться любой знак. В табл. 3 приведены примеры редактирования различных чисел с подавлением незначащих нулей различными символами-заполнителями, указанными в шаблоне (пробелом и знаком *).

Символы шаблона, отличные от символа выбора цифры, символа начала значимости и символа разделения полей, не замещаются цифрами данного. Они замещаются символом-заполнителем,

Таблица 3

Шаблон в шестнадцатеричном представлении	Исходное число	Отредактированное число на печати
4020202020202020	1234567C 0120406C 0001000C 0000001C	∪1 2 3 4 5 6 7 ∪∪1 2 0 4 0 6 ∪∪∪∪1 0 0 0 ∪∪∪∪∪∪1
5C20202020202020	1234567C 0012345C 0001000C 0000001C	*1 2 3 4 5 6 7 ***1 2 3 4 5 ****1 0 0 0 *****1

если значащая цифра еще не найдена, либо остаются без изменения, если значащая цифра уже встречалась. Эта особенность операции позволяет вставлять в редактируемое число нужные символы. Предположим, используется шаблон X'40206B2020206B202020', где X'6B' — значение запятой в коде ДКОИ. Результат редактирования по этому шаблону будет содержать в двух байтах (в тех, где в шаблоне содержится код X'6B') запятые. Однако если запятые расположены до значащих цифр, то они подавляются. Данные, которые приводились в табл. 3, можно отредактировать по этому шаблону. Результаты редактирования приведены в табл. 4.

Таблица 4

Шаблон в шестнадцатеричном представлении	Исходное число	Отредактированное число на печати
40206B2020206B202020	1234567C 0012345C 0001000C 0000001C	∪1, 2 3 4, 5 6 7 ∪∪∪∪1 2, 3 4 5 ∪∪∪∪∪t, 0 0 0 ∪∪∪∪∪∪∪∪1

Возможности команды ED не ограничиваются вставкой только запятых. С помощью этой команды можно вставлять любые знаки.

В этом примере можно заметить одну особенность: числа, которые содержат только дробную часть, отредактированы с подавлением запятой, отделяющей дробную часть от целой. Присутствие запятой можно учесть при редактировании, если в шаблоне вставить символ начала значимости. Этот символ замещается либо символом-заполнителем, либо цифрой из исходных данных так же, как и символ выбора цифры. Разница заключается в том, что после символа начала значимости в шаблоне операция редактирования продолжает выполняться так, будто в позиции редактируемого числа, соответствующем символу начала значимости, была значащая цифра. Это обозначает, что встречающиеся далее нули, даже если они незначущие, не подавляются, что возможно в связи с установкой необходимого значения в индикаторе значи-

мости. Например, если необходимо сохранить запятую, отделяющую в числе дробную часть от целой, и следующие за ней цифры, можно использовать следующий шаблон:

X'4020206B2020216B2020'. Если к тому времени, когда в шаблоне будет достигнут символ начала значимости (X'21'), не будет найдено ни одной значащей цифры, то индикатор значимости по этому символу устанавливается в 1, и все последующие цифры будут рассматриваться как значащие. Результаты редактирования рассматриваемых ранее чисел в этом случае будут такими, как показано в табл. 5 (сохраняется запятая и нули справа от нее).

Во всех приводимых ранее примерах код знака редактируемого числа игнорировался. Код знака «плюс» вызывает установку индикатора значимости в нуль, а код знака «минус» — в единицу. В приведенных примерах установка индикатора значимости в соответствии со знаком происходит уже после того, как закончен просмотр шаблона, поэтому на результате редактирования это не отражается

(когда шаблон кончается, редактирование исходного числа прекращается). Если бы в приводимых примерах какое-то число было отрицательным, то результат редактирования был бы точно таким же, как и для положительного числа.

Теперь предположим, что после просмотра исходного числа в шаблоне еще остались символы. Тогда значение индикатора значимости, установленное в результате анализа знака, будет продолжать управлять редактированием исходного данного. Конечно, после знака числа в исходной области нет больше цифр, которые нужно переслать в область результата. Поэтому желательно и в байты шаблона, которым уже нет соответствующих цифр в исходном данном, поместить не символы выбора цифр, а какие-либо символы. Эти символы либо останутся без изменения, если значение индикатора значимости равно 1, либо заменятся символом-заполнителем, если значение индикатора значимости равно 0. Таким образом, если редактируемое число отрицательное, его можно отметить, используя значение индикатора значимости.

Допустим, необходимо после отрицательных чисел через одну позицию печатать букву М, при этом незначащие нули чисел нужно подавить (значение буквы М в коде ДКОИ равно X'D4'). В табл. 6 приведены результаты выполнения операции редактирования по разным шаблонам отрицательных и положительных чисел. Знак «плюс» в десятичном числе имеет код шестнадцатеричной цифры С, знак «минус» — цифры D.

В первом шаблоне символом-заполнителем является пробел, во втором — знак *. Последние два байта обоих шаблонов не являются ни символом выбора цифры, ни символом начала значимости.

Таблица 5

Исходное число	Отредактированное число на печати
1234567C	01 2, 3 4 5, 6 7
0012345C	0012345
0001000C	0001000
0000001C	0000001,01

Шаблон в шестнадцатеричном представлении	Исходное число	Отредактированное число на печати
4020202020202040D4	1234567C 0098705D 0000001D	01 2 3 4 5 6 7 00 0009 8 7 0 5 0 M 00000001 0 M
5C20202020202040D4	1234567C 0098705D 0000001D	* 1 2 3 4 5 6 7 * * * * * 9 8 7 0 5 0 M * * * * * 1 0 M

ти, ни символом разделения полей. В этом случае байты в шаблоне заменяются символом-заполнителем (пробелом или *) для положительных чисел, так как знак плюс установит к этому времени значение индикатора значимости в 0, или остаются неизменными (пробел и буква M) для отрицательных чисел, так как знак минус установит к этому времени значение индикатора значимости в 1.

Команду ED можно использовать для того, чтобы отредактировать одной командой несколько данных. В этом случае используется символ разделения полей — X'22'. В шаблоне этот символ замещается символом-заполнителем, а индикатор значимости устанавливается в нуль. Следующие за ним символы и в шаблоне, и в исходных данных обрабатываются так же, как и для одного данного.

Предположим, имеется последовательность трех данных: длина первого данного 4 байта, второго — 3 байта, третьего — 5 байт. Первое данное необходимо разбить на группы по три цифры в каждой группе и разделить группы точками. Первое данное всегда предполагается положительным, поэтому обработка его знака не требуется. Дробную часть второго данного, состоящую из трех цифр, необходимо отделить запятой от целой части. Если второе данное содержит меньше 4 значащих цифр, то нужно оставить один нуль перед запятой. Если второе данное — положительное число, знак плюс нужно подавить, если — отрицательное, знак минус необходимо поместить справа от числа. Дробную часть третьего данного, состоящую из двух цифр, нужно запятой отделить от целой части. Кроме того, целую часть данного нужно разбить точками на группы по три цифры в группе. Если третье данное меньше 1, то число необходимо представить в виде дробной части со стоящей впереди запятой. Знак третьего данного обрабатывать не нужно. Между первым и вторым отредактированными числами должен быть по крайней мере один пробел, а между вторым и третьим — по крайней мере три пробела.

Для выполнения такого редактирования одной командой необходимо использовать следующий шаблон:

BD.DDD.DDDFSD,DDD—FBBDD.DDD.DDS,DD

Обозначения в шаблоне следующие:

B — пробел;

F — символ разделения полей;

D — символ выбора цифры;

S — символ начала значимости.

Замена первого символа разделения полей на символ-заполнитель обеспечит пробел между первым и вторым отредактированными числами. Символ начала значимости в части шаблона, которая соответствует второму данному, будет необходимым образом управлять редактированием величин, меньших единицы. Два пробела в начале части шаблона, соответствующей третьему данному, обеспечат два пробела между вторым и третьим отредактированными числами. Третий пробел между ними получается заменой символа разделения полей на символ-заполнитель. Пробелы внутри шаблона не рассматриваются как новые символы-заполнители, в качестве символа-заполнителя рассматривается только самый левый символ всего шаблона. Если область, где помещен шаблон, названа SHAB, а данные для редактирования — DATA, то по команде ED SHAB, DATA выполнится необходимое редактирование, и результат будет помещен в область SHAB. В табл. 7 приведены примеры исходных данных и результатов их редактирования по рассматриваемому шаблону.

Для редактирования данных можно использовать и команду EDMK (ОТРЕДАКТИРОВАТЬ И ОТМЕТИТЬ). Команда EDMK выполняется точно так же, как и команда ED (ОТРЕДАКТИРОВАТЬ), но дополнительно она помещает в общий регистр 1 адрес первой значащей цифры. Адрес заносится в разряды 8—31 этого регистра, разряды 0—7 не меняются. Если значимость устанавливается символом начала значимости в шаблоне, то адрес байта не заносится в регистр.

Таблица 7

Исходное число	Отредактированное число на печати
1234507C12305C123050789C	○1.234.507○12,305○○○○○1.230.507,89
0010009C00123C001000000C	○○○○○10.009○○○0,123○○○○○10.000,00
0004502C98007D000001210C	○○○○○4.502○98,007—○○○○○○○○○○○12,10
0000001C000001D000000001C	○○○○○○○○○○○1○○○0,001— ○○○○○○○○○○○○○○○,01

Перекодировка и команда ВЫПОЛНИТЬ. К командам преобразования данных из одного кода в другой относятся команды TR (ПЕРЕКОДИРОВАТЬ) и TRT (ПЕРЕКОДИРОВАТЬ И ПРОВЕРИТЬ).

Команды TR и TRT являются командами формата-SS. В них указываются два адреса памяти и используется общая длина для обоих операндов (длина первого операнда). Адрес первого операнда является адресом данных, которые нужно перекодировать. Адрес второго операнда указывает начало таблицы, которая ис-

пользуется для перекодировки. Байты, заданные адресом первого операнда, называются аргументами, байты в таблице — функциями.

Действие команды TR таково: из памяти выбирается байт-аргумент. Восьмиразрядный код этого байта, интерпретируемый как двоичное число, прибавляется к адресу второго операнда. Таким образом, получается адрес некоторого байта в таблице перекодировки. Извлекается содержимое этого байта и помещается на место байта-аргумента. Любые значения байта-аргумента и байта-функции являются допустимыми. Операция выполняется для всех байтов-аргументов, пока не будет исчерпан первый операнд.

Задача программиста состоит в составлении правильной таблицы перекодировки. Байт, который нужно перекодировать, можно рассматривать как индекс. Сумма его значения с начальным адресом таблицы задает адрес элемента в таблице. Элементы в таблице необходимо расположить так, чтобы по каждому полученному таким способом адресу в таблице находился байт, которым должен замещаться байт-аргумент.

Например, необходимо преобразовать буквы А, В, С в цифры 1, 2, 3 соответственно. Значения букв А, В, С в коде ДКОИ равны шестнадцатеричным числам X'C1', X'C2', X'C3' соответственно. Цифры 1, 2, 3 в коде ДКОИ имеют значения X'F1', X'F2', X'F3' соответственно. Предположим, таблицу перекодировки можно расположить в области TABL. Тогда в таблице байт с адресом TABL+X'C1' должен иметь значение X'F1', которым должна заместиться буква А. В байте с адресом TABL+X'C2' должно быть значение X'F2', а в байте с адресом TABL+X'C3' должно быть значение X'F3'. Если DATA — это имя области, где расположены буквы А, В, С, то команда TR DATA, TABL выполнит перекодировку букв в цифры. Действительно, если буква А встретится в качестве аргумента, ее значение X'C1' будет добавлено к адресу TABL, и значение байта с адресом TABL+X'C1' будет помещено на место буквы А. В байте с адресом TABL+X'C1' находится код цифры 1 (X'F1'). Аналогично буквы В и С будут заменяться цифрами 2 и 3.

Если в качестве аргумента может встретиться любая из возможных восьмиразрядных комбинаций нулей и единиц, то таблица перекодировки должна состоять из 256 байт. В отдельных случаях аргументом могут быть только некоторые коды байт, например, только латинские буквы от А до Z. Их значения в коде ДКОИ находятся в границах от X'C1' до X'E9' включительно. В этом случае достаточно подготовить только те байты таблицы, которые соответствуют значениям аргументов, а именно, байты с отисчисляемыми адресами от X'C1' до X'E9'. Остальные байты таблицы можно использовать в программе произвольным образом.

Команда TRT (ПЕРЕКОДИРОВАТЬ И ПРОВЕРИТЬ), как и команда TR, использует таблицу байтов-функций. Байт-аргумент применяется для получения адреса в таблице функций. Однако байт-аргумент в результате выполнения команды TRT не изменя-

ется. Байт-аргумент используется только для получения адреса байта-функции, а затем байт-функция только извлекается и используется для того, чтобы определить дальнейший ход выполнения операции. Если адресованный байт-функция равен нулю, то проверка данного байта-аргумента прекращается, выбирается и начинает проверяться следующий байт-аргумент. Если байт-функция не нуль, операция заканчивается, при этом адрес соответствующего аргумента помещается в общий регистр 1 (в младшие 24 разряда, старшие 8 разрядов не меняются), а байт-функция — в младшие 8 разрядов общего регистра 2 (разряды 0—23 регистра 2 не меняются). Байты первого операнда выбираются слева направо один за другим. Выборка функции выполняется таким же образом, как в команде TR. Если для всех просмотренных байт-аргументов не встретится ненулевой байт-функция, то выполнение команды заканчивается, а признак результата устанавливается равным 0. Если операция заканчивается при встрече ненулевого байта-функции (однако еще не все байты-аргументы были просмотрены), то признак результата устанавливается равным 1. Если же ненулевой байт-функция соответствует самому последнему байту-аргументу, то признак результата устанавливается равным 2.

С помощью команды TRT можно проверить некоторую последовательность байтов-аргументов, выбирая в ней те байты, которые необходимы для обработки, например ошибочные символы, пробелы, запятые и другие знаки, использующиеся как разделители. Рассмотрим следующий пример. Имеется массив записей, расположенных в памяти, начиная с адреса DAN. Записи в массиве друг от друга отделяются пробелами. Внутри записей пробелы отсутствуют. Длина каждой записи неизвестна, но она не более 256, включая пробел-разделитель. Необходимо выделить каждую запись из массива и поместить в область SAP для обработки.

Название	Операции	Операнды
...	LA	4,DAN
	LA	5,SAP
	TRT	Ø(256,4),TABL
	SR	1,4
	EX	1,PERES
PERES	MVC	Ø(Ø,5),Ø(4)
DAN	DC	X'56781234534Ø134Ø'
SAP	DS	CL256
TABL	DC	XL63'Ø'
	DC	X'1' байт для пробела
	DC	XL192'Ø'
...

Вместо того чтобы последовательно сравнивать каждый байт с пробелом для нахождения конца записи, найти пробел можно сразу с помощью одной команды TRT. Для этого необходимо по-

строить таблицу из 256 байт. Пробелу в коде ДКОИ соответствует шестнадцатеричное значение X'40', поэтому байт 64 от начала таблицы должен быть отличен от нуля, а остальные байты таблицы должны содержать нули. Команда TRT в этом случае прекратит свое выполнение при нахождении в исходном поле пробела. Команды, приведенные выше, выполняют выделение и пересылку самой первой записи.

Команды LA загружают начальный адрес первой записи (DAN) в регистр 4 и начальный адрес рабочей области для записи (SAP) в регистр 5. По команде TRT последовательно извлекаются байты первой записи (с адреса DAN начинается первая запись), содержимое каждого байта добавляется к адресу таблицы (TABL) и содержимое байта таблицы, адрес которого получается таким образом, анализируется на нуль. Если в байте исходной записи находится не пробел, то ему в таблице соответствует байт, содержимое которого равно нулю. Содержимое только одного байта таблицы, соответствующего пробелу, отлично от нуля. Поэтому, если в исходной записи встретится пробел, то выполнение операции прекратится, при этом в регистр 1 загрузится адрес байта в записи, содержащего пробел. Команда SR вычитает из полученного адреса начальный адрес записи. В результате в регистре 1 получается длина записи.

Пересылка записи выполняется с помощью команды EX (ВЫПОЛНИТЬ), которая ранее не рассматривалась. Команда EX, относящаяся к командам перехода, указывает, что необходимо выполнить команду, адрес которой указан во втором операнде команды EX. Команда по указанному адресу выполняется только после ее модификации содержимым общего регистра, заданного первым операндом в команде EX. Модификация выполняется путем логического сложения содержимого разрядов 8—15 команды, адрес которой указан в команде EX, с последними восемью разрядами общего регистра, указанного в команде EX. Логическое сложение не изменяет ни содержимое регистра, ни команду в памяти и производится только при выполнении команды. Признак результата устанавливает выполняемая команда. Эта возможность модифицировать команду, выполняемую по команде EX, позволяет косвенным путем задавать для нее длину, индекс, маску, непосредственный операнд.

В данном случае в команде EX косвенным путем задается длина для команды пересылки MVC. В команде MVC записаны два явных адреса: адрес пересылаемой записи и адрес рабочей области. Оба смещения равны нулю, так как базовые адреса сразу указывают на начало полей. Длина операндов, указанная в команде MVC, равна нулю. Но при выполнении команды MVC по команде EX длина будет установлена в результате сложения разрядов 8—15 команды MVC с содержимым регистра 1. В регистре 1 находится длина первой записи, поэтому команда MVC перешлет первую запись в рабочую область. Запись при выполнении приведенных команд будет пересылаться полностью, включая послед-

ний пробел-разделитель, так как значение длины, которое указывается в регистре 1, равно длине записи; а при выполнении команды MVC пересылается число байт на единицу больше заданной длины. Пробел, находящийся за записью, также пересылается вместе с ней.

3.4.6. Способы организации переходов в программе

Команды в программе обычно выполняются последовательно, в том порядке, в котором они записаны. Последовательное выполнение команд можно нарушить, если записать в программе какую-нибудь из команд переходов, которые могут передать управление командам, расположенным в разных частях программы. Кроме того, команды переходов позволяют обращаться к подпрограммам и организовывать циклы (неоднократное выполнение отдельных частей программы).

Команды переходов. В системе команд ЕС ЭВМ предусмотрены команды условного и безусловного переходов. При использовании команд безусловного перехода управление передается той команде, адрес которой указан в команде перехода. При выполнении команд условного перехода анализируется признак результата, устанавливаемый той или иной командой, и в зависимости от результата анализа либо происходит переход на указанную команду, либо выполняется команда, следующая за командой перехода.

Команды переходов являются командами формата RR, RX или RS. Первый операнд команд перехода определяет общий регистр или маску, указывающую проверяемый признак результата. Второй операнд указывает адрес команды, на которую выполняется переход.

Команды перехода, за исключением команды EX, которая тоже относится к командам переходов, не изменяют установленный признак результата. В результате выполнения команды EX признак результата устанавливается той командой, которая выполняется по указанию команды EX.

При выполнении команд переходов может возникнуть прерывание только при выполнении команды EX. Это может быть в следующих случаях:

- команда EX в свою очередь указывает на команду EX (прерывание из-за некорректности команды **ВЫПОЛНИТЬ**);

- команда EX указывает на команду, хотя бы одно полуслово которой выходит за пределы памяти, имеющейся в данной ЭВМ (прерывание по адресации);

- второй операнд команды EX определяет нечетный адрес команды, которая должна выполняться (прерывание по спецификации);

- ключ памяти операнда не совпадает с ключом защиты в слове состояния программы.

Команды условного перехода BC и BCR выполняют переход по адресу, указанному вторым операндом, только в том случае, если установленный признак результата соответствует коду, за-

данному в виде маски в первом операнде команды. В противном случае продолжается выполнение обычной последовательности команд в программе. Четыре разряда, отведенные в команде для первого операнда и используемые в качестве маски, соответствуют, слева направо, четырем значениям признака результата (0, 1, 2, 3). Соответствие между значениями маски и признаком результата приведено в табл. 8.

Таблица 8

Разряды команды перехода	Значение маски	Признак результата
8	1000	0
9	0100	1
10	0010	2
11	0001	3

Переход происходит, если устанавливается признак результата, соответствующий разряду маски, установленному в 1. Таким образом, выполнение команд BC и BCR зависит от признака результата, который устанавливается в результате выполнения арифметических операций, операций сравнения и сдвига, логических операций.

Для примера использования команд условного перехода рассмотрим следующую задачу. Допустим, даны три числа (A, B, C) с фиксированной точкой длиной в слово, которые могут иметь любой знак. Необходимо присвоить всем числам знак плюс и записать числа в память, расположив их по возрастанию. Поставленную задачу можно выполнить, написав следующую программу:

Название	Операция	Операнды	Название	Операция	Операнды
BEGIN	BALR	15,Ø	HA2	CR	2,3
	USING	*,15		BC	X'C',OUT
	LM	1,3,A		LR	6,3
	LPR	1,1		LR	3,2
	LPR	2,2		LR	2,6
	LPR	3,3		BC	15,HA1
HA1	CR	1,2	OUT	STM	1,3,A
	BC	12,HA2		SVC	14
	LR	6,1	A	DC	F'-1'
	LR	1,2	B	DC	F'12'
	LR	2,6	C	DC	F'3'
				END	BEGIN

По команде LM в регистры 1, 2, 3 загружаются числа A, B, C соответственно. Затем используется команда LPR (ЗАГРУЗКА ПОЛОЖИТЕЛЬНАЯ), которая помещает абсолютное значение второго операнда на место первого. При выполнении этой операции знак отрицательных чисел изменяется на противоположный, положительные числа остаются без изменения. Соответственно командой LNR (ЗАГРУЗКА ОТРИЦАТЕЛЬНАЯ) можно присваивать числам знак минус. Следующая команда CR (СРАВНЕНИЕ) — команда формата RR. Эта команда не меняет содержи-

мого регистров, она устанавливает признак результата равным нулю, если два операнда равны; единице, если первый операнд меньше второго; двум, если первый операнд больше второго (сравнение алгебраическое). Если бы один операнд находился не в регистре, а в памяти, то необходимо было бы использовать для сравнения команду С формата RX, а для сравнения чисел с фиксированной точкой длиной в полуслово — команду СН. По команде CR с именем HA1 сравнивается содержимое регистров 1 и 2. Следующая команда имеет маску 12 (в десятичной системе счисления), поэтому четыре разряда маски команды имеют двоичное значение 1100. При такой маске команда ВС проверяет, установлен ли признак результата равным 0 или 1, т. е. первый операнд равен второму или меньше второго. Если после выполнения команды CR установится признак 0 или 1, то произойдет переход по адресу HA2, в противном случае выполняется следующая команда. Таким образом, если число в регистре 1 меньше или равно числу в регистре 2, то будут пропущены три команды LR, выполняющие перестановку чисел в регистрах.

Команда CR с именем HA2 и следующая за ней команда ВС проверяют значения чисел в регистрах 2 и 3. В этой команде маска записана уже в виде числа X'C', которое представляет двоичное значение 1100. Если число в регистре 2 меньше или равно числу в регистре 3, то при выполнении программы будут пропущены команды LR, выполняющие перестановку чисел в регистрах 2 и 3.

После команд сравнения и обмена содержимого регистров 2 и 3 следует команда ВС с маской 15 (в десятичной системе счисления). Эта маска в разрядах команды представляется двоичным числом 1111. Любому признаку результата в команде ВС с такой маской соответствует 1, поэтому при выполнении этой команды всегда произойдет переход по адресу, указанному в команде перехода, т. е. команда с такой маской вызывает безусловный переход. И наоборот, команды ВС и BCR с маской 0000 никаких переходов вызывать не будут. В этом случае последовательное выполнение команд не будет изменяться, так как ни один из признаков результата не отмечен в маске. Таким образом, в приведенном примере команда BC 15, HA1 будет всегда вызывать переход по адресу HA1, т. е. если числа в регистрах 2 и 3 переставляются, то будет повторяться проверка чисел, находящихся в регистрах 1 и 2.

После того как числа в регистрах 1, 2 и 3 будут расположены по возрастанию, команда STM сохраняет их в памяти.

При необходимости выполнить операции сравнения над числами с плавающей точкой или десятичными числами можно применять аналогичные команды алгебраического сравнения с плавающей точкой или десятичной арифметики. В частности, для сравнения коротких чисел с плавающей точкой можно использовать команды CER и CE (СРАВНЕНИЕ КОРОТКОЕ), для сравнения длинных чисел с плавающей точкой — команды CDR и CD. Для десятичных чисел есть своя команда сравнения — CP (СРАВНЕНИЕ ДЕСЯТИЧНОЕ). При десятичном сравнении выполняется срав-

нение всех цифр и знаков справа налево, одновременно выполняется проверка на правильность кодов цифр и знаков. При выполнении операций сравнения над логическими данными можно применять команды CLR, CL, CLI, CLC (СРАВНЕНИЕ КОДОВ). Эти операции выполняют поразрядное сравнение обоих операндов слева направо, выполнение операции прекращается, как только встречаются несовпадающие разряды. Большим является число, содержащее единичный разряд. Во всех командах сравнения признак результата устанавливается в 0, 1, 2 соответственно, если операнды равны, первый операнд меньше второго, первый операнд больше второго. Команды переходов во всех случаях остаются прежними.

Командами перехода можно проверить признак результата, устанавливаемый командами, выполняющими арифметические операции. Эти команды устанавливают признак результата 0, 1, 2 или 3, если результат операции равен нулю, меньше нуля, больше нуля или возникло переполнение соответственно. Ниже приведен пример использования команды условного перехода после выполнения арифметической операции. Допустим, имеются два числа с фиксированной точкой. Необходимо получить их сумму и, если она не равна нулю, добавить к ней единицу. Это можно выполнить с помощью следующей программы:

Название	Операция	Операнды
BEGIN	BALR	15,0
	USING	*,15
	L	1,A
	A	1,B
	BC	8,HEDOB
	A	1,C
HEDOB	ST	1,D
	SVC	14
A	DC	F'3'
B	DC	F'2'
C	DC	F'1'
D	DS	F
	END	BEGIN

После выполнения команды сложения устанавливается признак результата, равный 0, если результат операции нуль, равный 1 или 2, если результат операции соответственно меньше или больше нуля. Маска, записанная в команде BC (1000), указывает, что переход нужно выполнить, если установится признак результата, равный 0. В этом случае будет выполняться переход по адресу HEDOB, а команда, добавляющая к сумме единицу, пропускается. Если признак результата будет равен 1 или 2, то команды будут выполняться последовательно, и к содержимому регистра 1 будет добавляться единица.

Команды условного перехода BC и BCR применяются также после таких логических команд, как TM (ПРОВЕРИТЬ ПО МАСКЕ) и TS (ПРОВЕРИТЬ И УСТАНОВИТЬ) формата SI. С помощью команды TM удобно проверять значения некоторых бит в байте. Команда TM проверяет состояние только тех разрядов в байте, которые указаны в маске, записываемой в самой команде. Разряды маски, равные 1, указывают на то, что соответствующий разряд байта памяти, адрес которого указывается в команде, выбирается для анализа. После выполнения команды TM устанавливаются следующие признаки результата:

0 — если все разряды байта, соответствующие единичным разрядам маски, равны нулю;

1 — если в разрядах байта, соответствующих единичным разрядам маски, есть и нули и единицы;

3 — если все разряды байта, соответствующие единичным разрядам маски, равны единицам.

Команды BC и BCR, выполняемые после команды TM, будут вызывать переход, если в четырехразрядной маске этих команд бит, соответствующий установленному командой TM признаку результата, равен 1. Например, если необходимо прервать последовательное выполнение команд и сделать переход на команду с именем ADR тогда, когда все биты байта с именем PEREC равны 1, то необходимо написать следующие команды:

Название	Операция	Операнды
	TM	PEREC, X'FF'
	BC	1, ADR

8 битов
mask

Команда TM анализирует все биты байта PEREC. Если они все равны 1, будет установлен признак результата, равный 3. Команда BC выполняет переход только тогда, когда признак результата равен 3, потому что маска 0001 соответствует этому значению. Если бы понадобился переход в случае только нулевых бит байта PEREC, в команде BC надо было бы написать маску 8 (1000).

Команда TS устанавливает признак результата исходя из значения крайнего левого разряда байта, адресованного в команде TS (команда TS формата SI, но при выполнении ее используется только адрес памяти, разряды непосредственного операнда не используются). Сразу после анализа нулевого разряда байта, указанного в команде, в байт заносятся все единицы. После этой команды может быть только два значения признака результата:

0 — крайний левый разряд указанного байта равен 0;

1 — крайний левый разряд указанного байта равен 1.

Установленный командой TS признак результата можно проверить командами BC и BCR, записав 1 в первом и (или) втором разряде маски.

Расширенные мнемонические коды. Как можно было заметить, записывать команды BC и BCR с маской довольно трудно. Необходимо четко представлять, какой признак результата устанавливает та или иная команда, чтобы правильно записать маску в команде перехода для анализа условия. Поэтому в язык Ассемблера введены расширенные мнемонические коды команд перехода, в которых не нужно указывать маску. По этим расширенным мнемоническим кодам транслятор построит машинную команду перехода BC или BCR с маской, которую надо было писать в команде программисту для определения условия перехода.

В табл. 9 приведены расширенные мнемонические коды, соответствующие маскам 0 и 15 (безусловный переход).

Таблица 9

Расширенная команда	Функция команды	Команда с маской
B D2(X2,B2)	БЕЗУСЛОВНЫЙ ПЕРЕХОД (формат RX)	BC 15,D2(X2,B2)
BR R2	БЕЗУСЛОВНЫЙ ПЕРЕХОД (формат RR)	BCR 15,R2
NOP D2(X2,B2)	НЕТ ОПЕРАЦИИ (формат RX)	BC Ø,D2(X2,B2)
NOPR R2	НЕТ ОПЕРАЦИИ (формат RR)	BCR Ø,R2

В приведенных командах формата RX используются явные адреса (B2 — регистр базы, X2 — регистр индекса, D2 — смещение). В этих командах может использоваться неявный адрес, например B NAME, B NAME(3) или NOP NAME, где NAME — символическое имя. В командах формата RR второй операнд R2 задает номер общего регистра, где должен находиться адрес перехода. Если, например, в программе необходимо выполнить безусловный переход на команду с именем PEREX, можно сделать это, записав вместо команды BC 15,PEREX команду B PEREX. Транслятор построит в машинной команде маску 15 исходя из мнемонического кода B.

Для выполнения переходов исходя из результата операций сравнения можно использовать расширенные мнемонические коды, приведенные в табл. 10.

Таблица 10

Расширенная команда	Функция команды	Команда с маской
BH D2(X2,B2)	ПЕРЕХОД ПО «БОЛЬШЕ»	BC 2,D2(X2,B2)
BL D2(X2,B2)	ПЕРЕХОД ПО «МЕНЬШЕ»	BC 4,D2(X2,B2)
BE D2(X2,B2)	ПЕРЕХОД ПО «РАВНО»	BC 8,D2(X2,B2)
BNH D2(X2,B2)	ПЕРЕХОД ПО «НЕ БОЛЬШЕ»	BC 13,D2(X2,B2)
BNL D2(X2,B2)	ПЕРЕХОД ПО «НЕ МЕНЬШЕ»	BC 11,D2(X2,B2)
BNE D2(X2,B2)	ПЕРЕХОД ПО «НЕ РАВНО»	BC 7,D2(X2,B2)

В командах записан явный адрес (обозначения, как в табл. 9), но можно использовать и неявный. В машинных командах, соответствующих записанным командам перехода по условию, транслятором будет построена такая маска, которая необходима для нужного перехода. Например, для команды BNE (ПЕРЕХОД ПО «НЕ РАВНО») построится маска 7 (в двоичном коде 0111), которая вызовет переход по указанному в команде адресу при всех значениях признака результата, кроме 0. Таким образом, переход выполнится во всех случаях неравенства операндов. Программа сравнения чисел А, В, С, рассматриваемая в 3.4.6, с использованием расширенных мнемонических кодов выглядит следующим образом:

Название	Операция	Операнды
BEGIN	BALR	15,0
	USING	*,15
	LM	1,3,A
	LPR	1,1
	LPR	2,2
HA1	LPR	3,3
	CR	1,2
	BNH	HA2 переход по не больше
	LR	6,1
	LR	1,2
HA2	LR	2,6
	CR	2,3
	BNH	OUT переход по не больше
	LR	6,3
	LR	3,2
OUT	LR	2,6
	B	HA1
	STM	1,3,A
A	SVC	14
	DC	F'-1'
B	DC	F'12'
C	DC	F'3'
	END	BEGIN

После арифметических команд для чисел с фиксированной точкой, чисел с плавающей точкой, десятичных чисел для выполнения переходов можно использовать расширенные мнемонические коды, приведенные в табл. 11.

Таблица 11

Расширенная команда	Функция команды	Команда с маской
BO D2(X2,B2)	ПЕРЕХОД ПО ПЕРЕПОЛНЕНИЮ	BC 1,D2(X2,B2)
BP D2(X2,B2)	ПЕРЕХОД ПО «+»	BC 2,D2(X2,B2)
BM D2(X2,B2)	ПЕРЕХОД ПО «-»	BC 4,D2(X2,B2)
BZ D2(X2,B2)	ПЕРЕХОД ПО «НУЛЮ»	BC 8,D2(X2,B2)
BNP D2(X2,B2)	ПЕРЕХОД ПО «НЕ+»	BC 13,D2(X2,B2)
BNM D2(X2,B2)	ПЕРЕХОД ПО «НЕ-»	BC 11,D2(X2,B2)
BNZ D2(X2,B2)	ПЕРЕХОД ПО «НЕ НУЛЬ»	BC 7,D2(X2,B2)

В командах записан явный адрес (обозначения, как в табл. 9), но можно использовать и неявный. Программа для приводимого в 3.4.6 примера сложения двух чисел с фиксированной точкой с использованием расширенного мнемонического кода выглядела бы следующим образом:

Название	Операция	Операнды
BEGIN	BALR	15,0
	USING	*,15
	L	1,A
	A	1,B
	BZ	HEDOB переход по нулю
	A	1,C
HEDOB	ST	1,D
	SVC	14
A	DC	F'3'
B	DC	F'2'
C	DC	F'1'
D	DS	F
	END	BEGIN

Расширенные мнемонические коды, приведенные в табл. 11, можно также использовать после команд загрузки, после которых вырабатывается признак результата, таких, как LTR, LCR, LPR, LNR, LTDR, LTER, LCDR, LCER, LPDR, LPER, LNDR, LNER. После выполнения этих команд устанавливается такой же признак результата, как и для арифметических операций. Если нужно проанализировать знак числа, находящегося в общем регистре, можно, например, использовать команду LTR (ЗАГРУЗКА И ПРОВЕРКА) формата RR и следующую за ней команду условного перехода BM или BP. По команде LTR содержимое второго регистра пересылается в первый регистр, при этом в зависимости от знака и величины второго операнда устанавливается следующий признак результата: 0 — если операнд равен нулю; 1 — если операнд меньше нуля; 2 — если операнд больше нуля.

Исходя из установленного признака результата будет выполнен тот или иной переход по команде условного перехода.

Команды, приведенные в табл. 11, можно использовать также после команд арифметического сдвига SRA, SLA, SLDA, SRDA, команд редактирования ED, EDMK, так как при выполнении этих команд тоже устанавливается признак результата, равный 0, 1, 2 или 3, если результат соответственно равен нулю, больше или меньше нуля или возникло переполнение. При выполнении логических команд типа X, O, N, а также после команды TS устанавливаются только два признака результата: 0 — результат равен нулю; 1 — результат не равен нулю. После выполнения этих логических команд можно использовать команды BZ и BNZ.

Как отмечалось раньше, после команды TM (ПРОВЕРИТЬ ПО МАСКЕ) установка признака результата несколько специфич-

на. Поэтому введены расширенные мнемонические коды, которые используются после команды ТМ. Эти коды приведены в табл. 12.

Таблица 12

Расширенная команда	Функция команд	Команда с маской
BO D2(X2,B2)	ПЕРЕХОД, ЕСЛИ ВСЕ 1	BC 1,D2(X2,B2)
BM D2(X2,B2)	ПЕРЕХОД, ЕСЛИ ЕСТЬ 1 И 0	BC 4,D2(X2,B2)
BZ D2(X2,B2)	ПЕРЕХОД, ЕСЛИ ВСЕ 0	BC 8,D2(X2,B2)
BNO D2(X2,B2)	ПЕРЕХОД, ЕСЛИ НЕ ВСЕ 1	BC 14,D2(X2,B2)

Обозначения в табл. 12 такие же, как в табл. 9. В командах можно использовать явный адрес. Например, если необходимо выполнить переход на команду с адресом ADR в случае, если все биты байта PEREC равны 1, то можно написать следующие команды:

Название	Операция	Операнды
	ТМ ВО	PEREC,X'FF' ADR

Во всех приводимых примерах с расширенными мнемоническими кодами можно было указывать регистр индекса. Например, в последнем случае можно было записать команду ВО ADR(2), используя неявный адрес с индексом. Тогда выполнялся бы переход по адресу ADR, увеличенному на содержимое регистра индекса 2.

Организация циклов. При программировании почти всегда возникает необходимость обеспечить неоднократное выполнение некоторых команд программы, т. е. организовать цикл. Цикл — это повторение выполнения некоторого участка программы заданное число раз. Каждое повторение может происходить с новыми значениями некоторых величин, участвующих в вычислениях и называемых параметрами цикла. При организации циклов, как правило, используются различные команды переходов. Рассмотрим возможности организации циклов и используемые при этом команды переходов на простых примерах.

Предположим, имеется 10 чисел с фиксированной точкой, каждое длиной в слово. Числа расположены последовательно в памяти, начиная с адреса TABL. Необходимо получить сумму этих 10 чисел и поместить ее по адресу SUM.

Для получения суммы в программе необходимо организовать цикл. Рассмотрим несколько способов реализации этой задачи. В любом случае в структуре цикла можно выделить следующие этапы:

- подготовка цикла;
- выполнение вычислений цикла, т. е. рабочая часть цикла;

в) модификация параметров, т. е. изменение величин, участвующих в вычислениях;

г) проверка на окончание цикла.

Для получения суммы будет использоваться команда сложения (рабочая часть цикла). Каждое повторное выполнение вычислений будем называть повторением цикла. В поставленной задаче при каждом повторении цикла должно добавляться следующее число. Таким образом, изменяемой величиной в вычислениях, т. е. параметром цикла, будет адрес добавляемого числа. Во всех приведенных ниже способах реализации цикла для модификации адреса в команде сложения будет использоваться регистр индекса. Однако можно модифицировать адрес, изменяя регистр базы.

Первый вариант программы подсчета суммы:

Название	Операция	Операнды
BEGIN	BALR	15,0
	USING	*,15
	SR	6,6 подготовка цикла
	SR	10,10
AD	A	6,TABL(10) вычисления
	A	10,C4 модификация
	C	10,C40 проверка на окончание цикла
	BL	AD
	ST	6,SUM
	SVC	14
TABL	DC	F'1,2,3,4,5,6,7,8,9,10'
SUM	DS	F
C4	DC	F'4'
C40	DC	F'40'
	END	BEGIN

В приведенной программе числа с фиксированной точкой, которые необходимо сложить, определены с помощью оператора DC (см. 3.6). Числа в памяти будут расположены последовательно, начиная с адреса TABL. Адрес каждого очередного числа будет на 4 байта больше, чем адрес предыдущего числа.

Регистр 6 в программе используется для накопления суммы, а регистр 10 как регистр индекса для модификации адреса. Необходимо, чтобы в исходном состоянии эти регистры содержали нули. Очистка регистров выполняется командами SR. Эти команды выполняют подготовку цикла.

Далее выполняется подсчет суммы. По команде с именем AD к содержимому регистра 6 прибавляется число из памяти. При первом выполнении команды адрес этого числа равен адресу TABL, так как регистр индекса 10 содержит нуль. Значит, первое исходное число, находящееся в памяти по адресу TABL, складывается с содержимым регистра 6, в котором находится нуль. Далее необходимо, чтобы во время следующего повторения цик-

ла к содержимому регистра 6 добавилось следующее число, адрес которого равен $TABL+4$. Для этого содержимое индексного регистра 10 следующей командой A увеличивается на 4. Последнее из десяти складываемых чисел имеет адрес $TABL+36$. Модификация регистра индекса выполняется перед проверкой на окончание цикла. После добавления последнего из исходных чисел, адрес которого равен $TABL+36$, регистр 10 будет содержать число 40.

Таким образом, когда в регистре 10 окажется число 40, следует прекратить повторение вычислений (закончить цикл) и продолжить последовательное выполнение команд. Проверка цикла на окончание в приведенной программе выполняется двумя командами: C и BL. Команда C сравнивает содержимое регистра индекса 10 с константой 40. Команда условного перехода BL (ПЕРЕХОД ПО «МЕНЬШЕ») проверяет результат сравнения. Если содержимое регистра индекса 10 меньше 40, то выполняется переход по адресу AD, в противном случае выполняется команда, следующая за BL. Переход на повторение цикла (по адресу AD) выполнится 9 раз, при этом команда сложения выполнится 10 раз. Нужно отметить, что в приведенной программе модификация параметра и проверка на окончание цикла выполнены двумя отдельными командами.

Ниже приведен второй вариант организации цикла:

Название	Операция	Операнды
BEGIN	BALR USING SR LA	15,0 *,15 6,6 подготовка цикла 10,36
AD	A S BNM ST SVC	6,TABL(10) вычисления 10,C4 модификация AD проверка на окончание 6,SUM 14
TABL SUM C4	DC DS DC END	F'1,2,3,4,5,6,7,8,9,10' F F'4' BEGIN

Во втором варианте число команд, повторяющихся в цикле, на одну меньше. Но эта команда выполнялась бы в цикле 10 раз, поэтому время работы данной программы будет меньше, чем предыдущей.

Исходное содержимое регистра индекса перед началом вычислений в цикле устанавливается равным 36, это число загружается в регистр 10 с помощью команды LA. Теперь при выполнении команды с именем AD в первый раз будет добавляться число, адрес которого $TABL+36$. Далее из содержимого регистра индекса 10 вычитается число 4, т. е. при следующем повторении цикла будет добавляться число с адресом $TABL+32$. Команда сравнения

здесь отсутствует. Сразу после вычитания числа 4 из содержимого индексного регистра 10 выполняется команда условного перехода BNM (ПЕРЕХОД ПО НЕ «—»). Эта команда использует признак результата после арифметической команды S. Пока индекс в регистре 10 будет больше или равен нулю, будет выполняться переход на команду AD, добавляющую следующее число, в противном случае цикл будет закончен. В то время, как в первом варианте программы в цикле выполняются 4 команды, во втором варианте выполняются только три команды.

Ниже приведен третий вариант программы, где в цикле выполняется минимальное число команд — две. Это достигается использованием команды BXLE (ПЕРЕХОД ПО ИНДЕКСУ МЕНЬШЕ ИЛИ РАВНО), которая представляет как бы сочетание команд СЛОЖЕНИЕ, СРАВНЕНИЕ И УСЛОВНЫЙ ПЕРЕХОД.

Название	Операция	Операнды
BEGIN	BALR	15,0
	USING	*,15
	SR	6,6 подготовка цикла
	SR	10,10
	LA	12,4
AD	LA	13,36
	A	6,TABL(10) вычисления
	BXLE	10,12,AD
	ST	6,SUM
	SVC	14
TABL	DC	F'1,2,3,4,5,6,7,8,9,10'
SUM	DS	F
	END	BEGIN

Команда BXLE имеет следующий вид: BXLE R1,R3,D2(B2), если адрес перехода задан явно, или BXLE R1,R3,A, если задан неявный адрес перехода (D2 — смещение, B2 — регистр базы, R1, R3 — номера общих регистров, A — адрес перехода). В регистре R1 находится первый операнд, значение которого увеличивается при выполнении команды. Приращение, т. е. число, на которое увеличивается первый операнд, должно находиться в регистре R3 и может быть положительным или отрицательным. Предельное значение, с которым производится сравнение нарастаемого содержимого R1, тоже должно находиться в общем регистре, но номер регистра явно в команде не указывается. Этот регистр всегда нечетный и выбирается следующим образом:

если регистр, указанный операндом R3, четный, то предельное значение должно быть в нечетном соседнем регистре с большим номером;

если регистр, указанный операндом R3, нечетный, то считается, что в команде BXLE используется один регистр и для приращения, и для предельного значения.

При выполнении команды **BXLE** приращение из регистра **R3** складывается с содержимым регистра **R1** и сумма алгебраически сравнивается с предельным значением. Затем сумма помещается в регистр **R1** независимо от того, происходит переход или нет. Если сумма меньше или равна предельному значению, выполняется переход по адресу, указанному в команде перехода. Если сумма больше предельного значения, выполняется обычная последовательность команд.

В приведенной программе имеются три регистра, назначение которых таково: регистр **10** содержит индекс; регистр **12** содержит число, на которое нужно увеличивать индекс после каждого повторения цикла, т. е. число **4**; регистр **13** содержит число **36**, которое является предельным значением. В начале программы эти регистры загружаются необходимыми первоначальными значениями.

Действие присутствующей в программе команды **BXLE 10,12,AD** будет состоять в следующем: содержимое регистра **12** (число **4**) прибавляется к содержимому регистра индекса **10**, содержащего вначале нуль. Если эта сумма меньше или равна содержимому регистра **13**, где находится максимально допустимое значение индекса (число **36**), происходит переход по команде с адресом **AD**, в противном случае выполняется команда, следующая за командой **BXLE**.

В тех случаях, когда приращение должно быть отрицательным (например, если нужно складывать числа в порядке, обратном тому, как они расположены в памяти), можно использовать команду **BXH (ПЕРЕХОД ПО ИНДЕКСУ БОЛЬШЕ)**. Эта команда подобна команде **BXLE**, за исключением того, что переход выполняется, если сумма больше заданного предельного значения, с которым выполняется сравнение.

Для организации циклов можно использовать команду **BSTR** формата **RR** или **BCT** формата **RX (ПЕРЕХОД ПО СЧЕТЧИКУ)**. В этих командах из содержимого регистра, указываемого в качестве первого операнда, алгебраически вычитается единица. Если результат равен нулю, продолжается последовательное выполнение команд; если результат не равен нулю, выполняется переход по адресу, указанному вторым операндом (в команде **BSTR** адрес перехода находится в регистре). Например, приводимую ранее программу сложения десяти чисел можно также написать с использованием команды **BCT (ПЕРЕХОД ПО СЧЕТЧИКУ)**, если выполнять модификацию параметра цикла независимо от проверки на окончание цикла. Ниже приведена программа, реализующая этот случай.

В программе командой **LA** в регистр **11** загружается число **10**, т. е. подготавливается счетчик числа повторений цикла. Команда **BCT** вычитает из счетчика единицу, и, пока он не станет равным нулю, передает управление команде с именем **AD**. Как только в регистре **11** будет **0** (после десяти повторений цикла), выполняется команда, следующая за командой **BCT**.

Название	Операция	Операнды
BEGIN	BALR USING SR SR LA	15,Ø *,15 6,6 подготовка цикла 1Ø,1Ø 11,1Ø
AD	A A BCT ST SVC	6,TABL(1Ø) вычисления 1Ø,C4 модификация 11,AD проверка на окончание 6,SUM 14
TABL	DC	F'1,2,3,4,5,6,7,8,9,1Ø'
SUM	DS	F
C4	DC	F'4'
	END	BEGIN

В командах BCT и BCTR в качестве адреса перехода может быть указан нуль. В этом случае выполняется только вычитание из счетчика единицы, а переход отсутствует.

Организация переходов с помощью переключателей. Для организации переходов в программе могут использоваться переключатели. Существует несколько способов составления переключателей. Например, для переключателя может быть использован какой-либо байт или несколько байт основной памяти. В зависимости от значения байта (или байт) выполняется переход на ту или иную часть программы.

Предположим, в программе необходимо различать, выполняется данная часть программы первый раз или второй, так как в первом случае после выполнения этой части программы необходимо выполнить одни действия, а во втором — другие. Реализация этой задачи приведена в следующей программе:

Название	Операция	Операнды
BEGIN	BALR USING MVI	15,Ø *,15 PEREC,X'ØØ' установка переключателя
* POVT	L L AR CLI BE S ST SVC	5,A 6,B 5,6 PEREC,X'ØØ' PERB проверка 5,D 5,RES2 14
PERB	A MVI	5,D PEREC,X'FF' изменение переключателя
*	ST B	5,B POVT

Название	Операция	Операнды
A	DC	F'20'
B	DC	F'40'
D	DC	F'10'
PEREC	DS	XL1
RES2	DS	F
	END	BEGIN

В начале программы в переключателе PEREC, для которого отведен один байт, устанавливается значение X'00'. В программе первый раз после получения суммы чисел с именами A и B добавляется число с именем D, и полученное число помещается на место числа с именем B. Второй раз вычисляется сумма числа с именем A и нового числа, находящегося по адресу B, а из полученной суммы вычитается число с именем D. После команд L, L и AR, выполняющих сложение чисел с именами A и B, стоят команды анализа переключателя (CLI, BE). Значение переключателя сравнивается со значением X'00'. Если значение переключателя равно X'00', выполняется переход на команду с именем PERB. В противном случае выполняется та часть программы, которая следует непосредственно за командами анализа переключателя. В части программы, начиная с команды с именем PERB, которой передается управление, если переключатель PEREC равен X'00', кроме необходимых вычислительных действий для первого случая, устанавливается новое значение переключателя X'FF'. Затем управление передается на повторение первоначальных вычислительных действий. После получения второй суммы чисел с именами A и B снова анализируется переключатель PEREC. Его значение будет не равно X'00', а поэтому управление передается другой части программы, которая запоминает результат в области RES2. Как можно было заметить, все переходы выполнялись командами условного или безусловного перехода с использованием расширенных мнемонических кодов.

В приведенной программе в качестве переключателя использовались все разряды байта. Значение такого переключателя изменяется командами пересылки, а проверяется командами сравнения. Но иногда в программе необходимо использовать несколько переключателей, и тогда отводить для каждого переключателя один байт нерационально. В таких случаях в качестве переключателей используются отдельные биты в байтах памяти. Каждый отдельный такой бит называется индикатором. Установка и проверка индикаторов выполняется с помощью логических команд типа O, N, X, TM, TS.

Предположим, выполняется анализ некоторого текста. Символы текста выбираются последовательно байт за байтом. Для цифры должна быть выполнена одна обработка, для букв — другая,

для специальных знаков — третья. Кроме того, существует одинаковая обработка для всех знаков. Для того чтобы осуществить переход к выполнению нужной обработки в зависимости от типа символа, можно ввести переключатель, в котором, например, бит 0 будет устанавливаться в 1, если символ — буква, бит 1 — если цифра, бит 2 — если специальный знак. Вначале весь переключатель устанавливается в нуль. Затем каждый рассматриваемый символ анализируется, и устанавливается в 1 бит, соответствующий символу. Допустим, рассматриваемый символ — цифра. Команда `OI PEREC,X'40` (`PEREC`—символическое имя переключателя) установит бит 1 переключателя, соответствующий цифре, в 1. Проверить состояние этого бита можно командой `TM PEREC,X'40'`.

Если некоторую обработку необходимо выполнить для буквы и цифры, но не для специального знака, то необходимо проанализировать два индикатора, соответствующие этим элементам. Команда `TM PEREC,X'CØ'` проверяет биты 0 и 1 переключателя, соответствующие цифрам и буквам. Записав за такой командой `TM` команду условного перехода `BM`, можно будет выполнить переход, если в проверяемых разрядах есть хотя бы одна единица, т. е. если была буква или цифра.

Погасить значение отдельного бита переключателя можно командой `NI`. Например, после конца обработки цифры можно командой `NI PEREC,X'BF'` установить в бите переключателя, соответствующем цифре, нуль. Команды типа `X` используются для переустановки значений бит при необходимости выполнять действия поочередно. Использование индикаторов значительно упрощает организацию переходов в программе.

3.4.7. Работа с подпрограммами

В программе могут быть такие последовательности команд, которые должны выполняться в программе несколько раз, причем в разных местах программы и с различными данными. Такую программу целесообразно составить так, чтобы повторяющаяся последовательность команд была оформлена в виде подпрограммы. Подпрограмма представляет собой отдельный блок программы, оформленный специальным образом, который может транслироваться вместе со всей программой (будем называть ее основной) или отдельно от нее. В любом случае программист должен на исходном языке программирования определить все связи между отдельными частями своей программы. Если подпрограмма транслируется отдельно от основной программы, то программист должен позаботиться и о том, чтобы к моменту, когда при выполнении программы потребуется обращение к подпрограмме, подпрограмма присутствовала в основной памяти. Это может быть обеспечено путем объединения основной программы и подпрограммы в одну выполняемую фазу при редактировании либо, если основная программа и подпрограмма при редактировании помещены в раз-

ные фазы, путем загрузки фазы с подпрограммой в основную память к моменту обращения к подпрограмме (об объединении см. 3.11). Загрузка выполняется с помощью операторов исходного языка. В дальнейшем будем считать, что подпрограмма находится в основной памяти вместе с основной программой.

Подпрограмма размещается в памяти, а затем используется каждый раз, когда потребуется выполняемая ею функция. При этом возникают вопросы: как организовать связь основной программы с подпрограммой, т. е. как перейти на подпрограмму и как возвратиться из подпрограммы на основную программу; каким образом происходит передача информации между основной программой и подпрограммой. Чтобы получить ясное представление, каким образом можно разрешить эти вопросы, рассмотрим простой пример. Допустим, имеются два числа, каждое число необходимо увеличить на 10, а затем удвоить. Конечно, все эти операции можно выполнить отдельно для каждого числа, но рассмотрим, как это можно сделать с использованием подпрограммы. Программа для такого вычисления может выглядеть следующим образом:

Название	Операция	Операнды
BEGIN	BALR	15,0
	USING	*,15
	L	14,ADPP загрузка адреса подпрограммы
*	L	3,FIRST
	BALR	13,14 обращение к подпрограмме
*	ST	3,RES1
	L	3,SECOND
	BALR	13,14
	ST	3,RES2
PEREP	SVC	14
ADPP	DC	A(PP)
FIRST	DC	F'1'
SECOND	DC	F'4'
RES1	DS	F
RES2	DS	F
* это конец основной программы		
PP	USING	*,14 подпрограмма
	A	3,DEC
	SLA	3,1
	BR	13 выход из подпрограммы
DEC	DC	F'10'
	END	BEGIN

Команды, выполняющие увеличение на 10 и удваивание числа, выделены в подпрограмму, которая транслируется вместе с основной программой. Начальный адрес подпрограммы — PP. Регистр 14 определяется в подпрограмме как регистр базы. Загрузка его выполняется в основной программе. В подпрограмме могут использоваться те же общие регистры, что и в основной про-

грамме. Тогда программист должен предусмотреть возможность сохранения этих регистров перед обращением к подпрограмме и восстановления их после выполнения подпрограммы. Приведенная программа с подпрограммой проста и в ней не рассматривается такой случай, но об этом необходимо помнить при составлении программ и использовании подпрограмм.

Связь между основной программой и подпрограммой осуществляется в приведенном примере с помощью команды BALR (ПЕРЕХОД С ВОЗВРАТОМ), являющейся командой безусловно-го перехода. Рассмотрим организацию обращения к подпрограмме на примере обработки одного числа, потому что для обработки другого обращение выполняется аналогично.

Используемая команда BALR является командой формата RR. По этой команде выполняется переход по адресу, содержащемуся в регистре, номер которого задается вторым операндом. Перед выполнением перехода в регистр, указанный первым операндом, помещается адрес команды, следующей за командой BALR (адрес возврата). В приведенной программе в регистр 14, используемый в команде BALR для адреса перехода, загружается адрес первой команды подпрограммы. Для сохранения адреса возврата в основную программу используется регистр 13. Таким образом, по команде BALR выполнится безусловный переход на первую команду подпрограммы, а в регистре 13 сохранится адрес возврата. В каждом случае из двух, когда команда BALR используется в программе, в регистре 13 будет свой адрес возврата. Для перехода к подпрограмме можно было использовать и команду BAL (ПЕРЕХОД С ВОЗВРАТОМ) формата RX. В этом случае адрес подпрограммы записывался бы прямо в команде: BAL 13,PP. Но при этом необходимо проследить, чтобы для базирования этого адреса (он неявный) существовал доступный регистр базы. В приведенной программе в этом случае до команды BAL 13,PP необходимо было бы поместить команду USING PP,14, а команду USING *, 14 из подпрограммы убрать.

После выполнения команд подпрограммы безусловный переход по адресу в регистре 13 (команда BR) обеспечивает выход в то место основной программы, откуда выполнялось обращение к подпрограмме, так как после каждого обращения к подпрограмме в регистре 13 будет необходимый адрес возврата.

Выход из подпрограммы можно организовать в разные точки основной программы исходя из результатов работы подпрограммы. В этом случае для организации выхода можно использовать команду условного перехода. Например, при выполнении сдвига в приводимой подпрограмме может возникнуть переполнение, если исходное число очень велико. Чтобы учесть такой случай, в подпрограмме можно вместо одной команды выхода из подпрограммы поставить две следующие команды:

Название	Операция	Операнды
	BO BR	PEREP 13

В случае переполнения выход из подпрограммы всегда будет выполняться на конец основной программы, а при нормальном результате будет происходить возврат в основную программу к той команде, которая следует за командой обращения к подпрограмме.

Рассмотрим, как происходит передача информации от основной программы к подпрограмме. Это можно сделать разными способами. В приведенной программе информация передается в регистре 3. Зная, что подпрограмма выполняет действия с содержимым регистра 3, перед обращением к подпрограмме в этот регистр помещается число, над которым будут производиться действия. Результаты работы подпрограммы основная программа выбирает тоже из регистра 3. При более сложных действиях, выполняемых подпрограммами, можно использовать для передачи информации несколько регистров или указать в регистре адрес поля, в котором содержится передаваемая информация.

Передачу информации можно осуществить и другим способом. Информация располагается сразу после команды BALR. Подпрограмма строится таким образом, что, используя регистр возврата, находит эти исходные данные или помещает свои результаты сразу за командой BALR. В этом случае необходимо по-другому организовать выход из подпрограммы. Рассматриваемую ранее программу при таком способе передачи информации можно записать следующим образом:

Название	Операция	Операнды
BEGIN	BALR USING L' BALR DC DS BALR DC DS SVC DC	15,Ø *,15 14,ADPP 13,14 H'1' H 13,14 H'4' H 14 A(PP)
ADPP		
* это конец основной программы		
PP	LH AH SLA STH B DC END	3,DEC подпрограмма 3,Ø(Ø,13) 3,1 3,2(Ø,13) 4(Ø,13) H'1Ø' BEGIN
DEC		

В этой программе обрабатываются числа с фиксированной точкой длиной в полуслово.

Рассмотрим первое обращение к подпрограмме (следующее выполняется аналогично). В регистре 13 после выполнения команды BALR, используемой для перехода к подпрограмме, будет находиться адрес следующей команды — адрес числа Н'1'. В подпрограмме в командах АН, STН, В записан явный адрес с использованием регистра 13 как регистра базы. В адресе в команде АН смещение равно нулю, индекс равен нулю, а базовый адрес, содержащийся в регистре базы 13, является адресом исходного числа, над которым необходимо выполнить операцию. По команде АН будет выполнено сложение содержимого регистра 3, куда загружено число 10, и исходного числа. Команда STН будет помещать результаты в область памяти, которая следует в основной программе через два байта после команды BALR. В регистре 13 содержится адрес байта, непосредственно следующего за командой BALR. Поэтому адрес области памяти, куда засылается результат, получается в результате сложения содержимого регистра 13 и смещения, равного 2. Команда В 4(Ø,13), в которой указан явный адрес для безусловного перехода, обеспечивает выход из подпрограммы не на следующую за BALR команду в основной программе, а на команду, имеющую адрес на 4 байта больше, чем адрес возврата в регистре 13. В результате использования такого адреса данные, расположенные за командой BALR, будут при выходе из подпрограммы пропущены.

Если используемых подпрограммой данных много, то можно задать только адрес этих данных в виде адресной константы, используя приведенный выше способ.

При программировании на языке Ассемблера в ДОС ЕС для установления связей между основной программой и подпрограммой можно использовать макрокоманды (см. 4.1):

CALL — обратиться к подпрограмме;

SAVE — запомнить регистры;

RETURN — вернуться к основной программе.

Макрокоманда CALL используется в основной программе для передачи управления подпрограмме, а также для передачи информации от основной программы к подпрограмме. Макрокоманда SAVE предназначена для запоминания содержимого общих регистров основной программы, используемых в подпрограмме. Макрокоманда RETURN используется в подпрограмме для того, чтобы после завершения работы подпрограммы передать управление основной программе и восстановить содержимое общих регистров основной программы. При использовании этих макрокоманд необходимо соблюдать правила использования регистров 0, 1, 13, 14, 15, которые называются регистрами связи. Регистры 0 и 1 предназначены для обмена информацией между подпрограммой и основной программой. Если передается адрес области, содержащей указанную информацию, то он должен быть помещен в регистр 1. Регистр 13

используется для передачи подпрограмме адреса области сохранения. Область сохранения — участок основной памяти, в которой должно быть произведено запоминание содержимого общих регистров, используемых в основной программе. Регистр 14 используется для перехода по адресу входа в подпрограмму.

Если подпрограмма транслируется отдельно от основной программы, то при организации перехода к подпрограмме и выхода из нее нужно подготовить информацию для Редактора с помощью команд, секционирования и соединения (см. 3.11).

3.5. РАБОТА СО ЗНАЧЕНИЕМ СЧЕТЧИКА АДРЕСА

Во время трансляции каждой машинной команде или области данных присваивается значение счетчика адреса. Если оператор назван именем, то значением этого имени является значение счетчика адреса, присвоенное этой команде. Программист в своей программе может воспользоваться текущим значением счетчика адреса, т. е. использовать терм — значение счетчика адреса, который записывается в виде знака *. Использовать знак * в операнде машинной команды — это то же самое, что поместить имя в поле названия оператора, а затем использовать это имя в операнде оператора. Использование значения счетчика адреса можно было увидеть в примерах, приводимых ранее. В операторе USING выражение, указывающее содержимое регистра базы, записывалось почти всегда с использованием термина *. Программист может управлять установкой счетчика адреса: определять его первоначальное значение, изменять его значение в программе, устанавливать значение счетчика адреса на определенную границу. Для этого программисту предоставлены операторы START, ORG, CNOP.

Допустим, программисту необходимо, чтобы его программа в результате трансляции располагалась с адреса X'4000'. Это он может сделать, записав в программе оператор START, который устанавливает первоначальное значение счетчика адреса.

Оператор START имеет следующий формат:

Название	Операция	Операнды
Любое символическое имя или пробел	<u>START</u>	Самоопределенный терм или пробел

Значение самоопределенного термина, записанного в поле операндов, будет начальным значением счетчика адреса для данной программы. Это значение должно быть кратным 8. Значения всех остальных адресов в программе устанавливаются относительно начального значения счетчика адреса. Таким образом, чтобы установить первоначальное значение счетчика адреса, равное X'4000',

программисту необходимо записать в начале своей программы следующий оператор:

Название	Операция	Операнды
NAME	START	X'4000'

Если в операторе START, присутствующем в программе, в поле операндов указан пробел, то транслятор установит начальное значение счетчика адреса в нуль.

В программе оператору START не должен предшествовать никакой оператор языка Ассемблера, который влияет на значение счетчика адреса. Например, если перед оператором START будет записана любая машинная команда, то такая последовательность команд будет неправильной. Если же перед оператором START будут находиться такие операторы, как ICTL, REPRO, PUNCH и комментарии, которые не влияют на значение счетчика адреса, то такая последовательность команд в программе будет правильной.

С помощью оператора START значение счетчика адреса удобно устанавливать равным адресу загрузки программы. В этом случае облегчается работа с распечаткой протранслированной программы во время отладки и анализа программы. Например, адрес программного сбоя прямо указывает ту команду, при выполнении которой он произошел.

Если в программе оператор START отсутствует, то начальное значение счетчика адреса будет нулевым.

Команда Ассемблера ORG дает возможность программисту изменять во время трансляции значение счетчика адреса.

Оператор ORG имеет следующий формат:

Название	Операция	Операнды
Символическое имя перехода или пробел	ORG	Простое переместимое выражение или пробел

Простое переместимое выражение в поле операндов оператора ORG указывает новое значение счетчика адреса. Любые символические имена в этом выражении должны быть предварительно определены, т. е. должны встретиться в поле названия каких-либо операторов до оператора ORG.

Пробел в поле операндов оператора ORG обозначает, что счетчику адреса присваивается значение, на единицу больше максимального адреса, использованного в программе (точнее, в программной секции) до данного оператора ORG. Использование оператора ORG в многосекционных программах рассматривается в 3.11.

Рассмотрим следующую программу, в которой используется оператор ORG:

Название	Операция	Операнды	Название	Операция	Операнды
NAME1	START BALR USING B	X'40000' 15,0 *,15 NAME2	NAME4	UNPK MVC B	BL1,OBL BL3,BL1 NAME5
OBL	DS	CL10	IND1	ORG DS	OBL CL2
BL1	DS	CL30	FILD1	DS	CL4
BL2	DS	CL2	FILD2	DS	CL46
BL3	DS	CL10		ORG	
NAME2	LA	1,100	NAME5	MVI L	IND1,X'FF' 10,FILD1
NAME3	MVI MVI	BL2,X'000' BL2+1,X'FF'	DAN	DC	PL10'1234567890'
PEREC	MVC	OBL,DAN		END	

В начале приведенной программы резервируется рабочая область памяти, которая разделена на поля OBL, BL1, BL2, BL3. Длина рабочей области равна 52 байтам. Имена полей и характеристики длины этих имен используются затем в машинных командах. Например, в команде MVC с именем PEREC не указана явная длина, число пересылаемых байт будет определяться неявной длиной первого операнда, которая равна 10 — характеристике длины имени OBL. Позже в программе возникает необходимость использовать эту же область памяти, но с другим разделением на поля. Тогда для того чтобы указать транслятору Ассемблера, что используется та же рабочая область OBL, оператором ORG значение счетчика адреса устанавливается равным значению имени OBL (начальному адресу рабочей области). Затем операторами DS определяются поля новых размеров, и им присваиваются имена (IND1, FILD1, FILD2). После перераспределения памяти оператор ORG с пробелом в поле операндов устанавливает в счетчике адреса значение, которое на единицу больше максимального значения адреса, использованного до этого в программе, т. е. значение, равное NAME4+4. Это значение будет присвоено имени NAME5.

Если бы до появления оператора ORG с пустым операндом значение счетчика адреса увеличилось так, что стало бы больше значения NAME4+4, то оператором ORG с пустым операндом в счетчике адреса установилось бы это максимальное значение.

Оператор ORG изменяет значение счетчика адреса во время трансляции и это влияет только на то, какие значения получают при трансляции символические имена. Например, после трансляции программы имена OBL и IND1 получают одинаковое значение, но будут иметь разную характеристику длины.

Многие машинные команды используют адреса памяти, которые должны быть расположены с целочисленной границы полусло-

на, слова или двойного слова (значение адреса должно быть соответственно кратно 2, 4 и 8). Например, адрес памяти, определяемый в командах с фиксированной точкой, должен быть на границе слова или полуслова. Процесс установки значения счетчика адреса на нужную границу (полуслова, слова, двойного слова) называется выравниванием на границу (полуслова, слова, двойного слова). В некоторых случаях выравнивание на нужную границу производится транслятором автоматически, но при необходимости можно произвести вынужденное выравнивание с помощью операторов CNOP, DC или DS. О выравнивании с помощью операторов DC или DS сказано в 3.6. Можно только отметить, что при выполнении выравнивания с помощью операторов DC или DS байты, пропущенные при выравнивании, либо заполняются нулями, либо пропускаются. В случае, если на необходимую границу должна быть помещена команда, такое выравнивание вызовет появление байтов, которые нарушают последовательность команд. В этом случае можно выполнить выравнивание с помощью оператора CNOP, так как в пропускаемых байтах по этой команде строятся команды безусловного перехода.

По оператору CNOP производится выравнивание значения счетчика адреса на границу полуслова, слова и двойного слова. Если счетчик адреса уже установлен на необходимую границу, то команда CNOP никакого действия не вызывает. Если указанное выравнивание требует увеличения значения счетчика адреса, то транслятор вырабатывает от одной до трех команд NOPR (НЕТ ОПЕРАЦИИ), каждая из которых занимает два байта.

Оператор CNOP имеет следующий формат:

Название	Операция	Операнды
Символическое имя перехода или пробел	CNOP	Два абсолютных выражения в форме: a,b

Символические имена, используемые в выражениях в поле операндов, должны быть предварительно определены. Операнд a указывает, на какой байт слова или двойного слова должен быть установлен счетчик адреса. Значениями операнда a могут быть 0, 2, 4 и 6. Операнд b указывает, находится ли байт a в слове (b=4) или в двойном слове (b=8). Правильные пары значений a и b и границы, на которые устанавливается счетчик адреса, приведены в табл. 13.

Оператор CNOP с другими значениями операндов a и b будет ошибочным.

Если в программе встречается оператор CNOP, а счетчик адреса находится на границе нечетного байта, то перед обработкой оператора CNOP транслятор сам выполняет предварительное выравнивание на границу полуслова.

Значение а, в	Граница
0,4	Начало слова
2,4	Середина слова (начало второго полуслова в слове)
0,8	Начало двойного слова
2,8	Второе полуслово двойного слова
4,8	Середина (третье полуслово) двойного слова
6,8	Четвертое полуслово двойного слова

Предположим, что текущее значение счетчика адреса находится на границе двойного слова (значение кратно восьми). Тогда оператор CNOP 0,8 не производит никакого действия. Теперь допустим, что имеется такая последовательность операторов:

Название	Операция	Операнды
	CNOP	6,8
	BALR	2,14

Если перед обработкой данных операторов значение счетчика адреса находится на границе двойного слова, то оператор CNOP вызовет построение трех команд безусловного перехода (NOPR), а команда BALR будет размещена в последнем полуслове двойного слова. В результате будут построены следующие команды:

Название	Операция	Операнды
	BCR	0,0
	BCR	0,0
	BCR	0,0
	BALR	2,14

Рассмотрим более конкретный случай использования оператора CNOP. В 3.4.7 приводился пример организации связи с подпрограммой, когда исходные данные для подпрограммы размещались сразу же после команды BALR. Данные в примере определялись оператором DC в виде чисел с фиксированной точкой длиной в полуслово. Эти числа размещаются транслятором на границе полуслова и в программе следуют непосредственно за командой BALR.

Допустим, надо таким же образом передать подпрограмме информацию, которая представляет собой числа с фиксированной точкой длиной в слово. Эти числа должны быть расположены на границе слова, так как этого требуют машинные команды, выполняющие операции над числами с фиксированной точкой. Опреде-

лечь эти числа и разместить их на границе слова можно оператором DC. Но вследствие выравнивания между командой BALR и константой могут появиться пропущенные байты: команда BALR занимает 2 байта, и если она расположена на границе слова, то до следующей границы слова, начиная с которой располагается следующее за командой BALR число, будет пропущено 2 байта. Если же команда BALR будет расположена на границе полуслова, то числа будут следовать непосредственно за командой. Чтобы команда BALR и числа всегда непрерывно следовали друг за другом, необходимо перед командой BALR поместить оператор CNOP 2,4. Этот оператор установит значение счетчика адреса на начало второго полуслова в слове. Начиная с этого адреса, будет размещаться команда BALR. Таким образом, следующая за командой BALR константа будет всегда размещена на границе слова, а между командой BALR и константой никогда не будет пропущенных байт. Если при размещении команды BALR в середине слова счетчик адреса увеличивается, то в пропускаемые байты помещается команда NOPR. Последовательность команд в программе прерываться не будет.

Рассматриваемая в 3.4.7 программа, выглядела бы таким образом:

Название	Операция	Операнды
BEGIN	BALR	15,0
	USING	*,15
	L	14,ADPP
	CNOP	2,4
	BALR	13,14
	DC	F'1'
	DS	F
	CNOP	2,4
	BALR	13,14
	DC	F'4'
	DS	F
	SVC	14
ADPP	DC	A(PP)
* конец основной программы		
PP	L	3,DEC
	A	3,0(0,13)
	SLA	3,1
	ST	3,4(0,13)
	B	8(0,13)
DEC	DC	F'10'
	END	BEGIN

3.6. ОПРЕДЕЛЕНИЕ И ИСПОЛЬЗОВАНИЕ ДАННЫХ

При написании программ программист использует множество исходных данных, над которыми выполняются различные операции. Выполнение операций программист определяет, записывая

машинные команды. Необходимо еще определить в программе те данные, над которыми будут производиться операции. Исходные данные — это либо информация, вводимая с внешних устройств, либо константы, определяемые в исходной программе. Для определения констант язык Ассемблера предоставляет программисту команду Ассемблера DC, которая позволяет ему легко определять все необходимые программе константы. Используя этот оператор, программист только записывает константы в понятной для него форме, а перевод их в двоичное представление на машинном языке выполняет транслятор.

3.6.1. Оператор определения констант

Так как система команд ЕС ЭВМ допускает выполнение действий над разными типами данных, то оператор DC позволяет соответственно определять различные типы констант: с фиксированной точкой, с плавающей точкой, десятичные, знаковые, двоичные, шестнадцатеричные и адресные. Адресные константы — это константы, значениями которых являются адреса памяти; они введены потому, что во многих случаях необходимо выполнять некоторые действия над адресами памяти. Одним оператором DC можно определить одну или несколько констант. Таким образом, программисту нет необходимости писать для каждой константы оператор DC.

Оператор DC имеет следующий формат:

Название	Операции	Операнды
Любое символическое имя, или пробел	DC	Один или несколько операндов, разделенных запятыми, в формате, описанном ниже

Если в поле названия оператора DC присутствует символическое имя, то оно является именем первой (или единственной) константы, определяемой этим оператором. Значение имени — это адрес константы. Характеристика длины этого имени равна длине в байтах первой (или единственной) константы, определяемой оператором DC.

Каждый операнд оператора DC включает: кратность, модификаторы, тип, константу (или константы). Кратность и модификаторы могут отсутствовать в операнде, но тип и сама константа (или константы) должны указываться обязательно. Для большинства типов констант в операнде можно записывать несколько констант. В этом случае указанные кратность, тип, модификаторы относятся ко всем константам в операнде.

Если в операнде указывается кратность, то константа или константы, записанные в операнде, создаются транслятором столько раз, сколько указывается значением кратности.

Тип константы, записанный в операнде оператора DC, указы-

бает транслятору, в какой машинный формат он должен переводить константу (или константы). При программировании важно определить константу сразу в таком виде, в котором она необходима для выполнения операции. Конечно, всегда можно перекодировать данные, но для перекодировок необходимо дополнительное время. Язык Ассемблера предоставляет программисту достаточно широкие возможности определения констант, допуская запись в операторе DC 13 типов констант (табл. 14).

Таблица 14

Тип	Константа	Машинный эквивалент
C	Знаковая	Двоичный восьмиразрядный код для каждого знака
X	Шестнадцатеричная	Двоичный четырехразрядный код для каждой шестнадцатеричной цифры
B	Двоичная	Двоичный формат (один бит соответствует одной цифре константы)
F	С фиксированной точкой	Число с фиксированной точкой, обычно слово
H	С фиксированной точкой	Число с фиксированной точкой, обычно полуслово
E	С плавающей точкой	Короткое число с плавающей точкой, обычно слово
D	С плавающей точкой	Длинное число с плавающей точкой, обычно двойное слово
R	Упакованная десятичная	Упакованный десятичный формат
Z	Распакованная десятичная	Десятичный формат с зоной
A	Действительная адресная	Значение адреса, обычно слово
Y	Действительная адресная	Значение адреса, обычно полуслово
S	Явная адресная	В виде регистра базы и смещения, полуслово
V	Внешняя адресная	Область, резервируемая для адреса внешнего имени, обычно слово

Модификаторы описывают константу (или константы). Можно описывать константу, указывая три модификатора: модификатор длины, модификатор масштаба, модификатор порядка. Модификатор длины может указываться для констант любого типа, модификаторы масштаба и порядка — только для констант с фиксированной и плавающей точкой.

Правила записи одного операнда оператора DC

Элементы операнда должны записываться обязательно в таком порядке: кратность, тип, модификаторы, константа (или константы).

Внутри операнда не должно быть пробелов, если это не знак в знаковой константе или в знаковом самоопределенном терме.

Записываемая в операнде константа (или константы) заключается в апострофы или в скобки.

При определении нескольких констант в операнде записываемые константы разделяются запятыми.

Кратность записывается либо десятичным самоопределенным термом без знака, либо абсолютным положительным выражением, заключенным в скобки.

Тип константы указывается одной из букв, приведенных в табл. 14.

Если в операнде указываются модификаторы, то они должны записываться в следующем порядке: длина, масштаб, порядок.

Модификатор длины записывается как L_p , где p — либо десятичный самоопределенный терм без знака, либо положительное абсолютное выражение, заключенное в скобки.

Модификаторы масштаба и порядка записываются соответственно как S_p и E_p , где p — либо десятичный самоопределенный терм, перед которым может быть записан знак, либо положительное абсолютное выражение, заключенное в скобки. Перед скобками может быть записан знак.

Все символические имена, используемые при записи выражений в кратности или модификаторах, должны быть предварительно определены (появиться в поле названия каких-либо операторов до данного оператора DC).

Транслятор Ассемблера F допускает запись в поле операндов оператора DC нескольких операндов. В этом случае операнды между собой разделяются запятыми, а каждый операнд записывается в таком же формате, как описывалось выше. Кроме того, транслятор Ассемблера F допускает указание модификатора длины в битах. Модификатор длины в битах записывается как L_p , где p — десятичный самоопределенный терм без знака, либо абсолютное выражение, заключенное в скобки.

При трансляции некоторые типы констант размещаются транслятором на границе полуслова, слова или двойного слова. Выравнивание зависит от типа константы и наличия модификатора длины.

Модификатор длины определяет количество байт памяти, отводимых для константы, и указывает явную длину константы. В случае присутствия в операторе DC модификатора длины выравнивание не производится. Если длина константы задана неявно (модификатор длины не указан), то для следующих типов констант производится выравнивание на границу:

- F — на границу слова;
- H — на границу полуслова;
- E — на границу слова;
- D — на границу двойного слова;
- A — на границу слова;
- Y — на границу полуслова;
- S — на границу полуслова;
- V — на границу слова.

Байты, пропущенные при выравнивании по оператору DC, заполняются нулями и не рассматриваются как часть константы. Если операнд определяет несколько констант, выравнивание производится только для первой константы.

3.6.2. Примеры определения и использования констант

Знаковая константа. Знаковую константу (тип C) можно использовать для задания в программе текстовой (символьной) информации. Например, если необходимо вывести на печать некоторое сообщение, то текст этого сообщения в программе может быть определен знаковой константой.

Знаковая константа записывается как последовательность знаков, заключенная в апострофы. В знаковой константе может быть записан любой знак кода ДКОИ. В одном операнде оператора DC может быть указана одна знаковая константа, каждый знак константы транслируется в один байт. Выравнивание на границу для знаковых констант не производится.

Максимальная длина константы (явная или неявная) — 256 байт. Если модификатор длины не указан, длина знаковой константы в байтах равна числу знаков, записанных в константе. Если указан модификатор длины, результат трансляции определяется следующим образом:

если число знаков в константе превышает указанную длину, то отбрасываются самые правые знаки;

если число знаков в константе меньше указанной длины, то константа справа дополняется пробелами.

При записи знаковой константы необходимо учитывать правило записи апострофа и знака &. Каждый апостроф или знак &, определяемые в знаковой константе, должны записываться двумя апострофами или двумя знаками &. В машинной программе появится только один апостроф или знак &. Ниже приводятся примеры записи знаковой константы.

Название	Операция	Операнды
NAME	DC	'С'ВЫЧИСЛЕНИЕ ВЫРАЖЕНИЙ'
NAME1	DC	CL25'ПРОГРАММА'
NAME2	DC	'С'ИМЯ TO&&RR'
NAME3	DC	2CL4'ABCDEF'
NAME4	DC	'С'ИМЯ"DØ1Ø1'

По оператору DC с именем NAME транслятор создаст в памяти 20 байт данных, каждый из которых соответствует знаку, записанному в константе. По оператору с именем NAME1 будет создана константа длиной 25 байт, так как модификатор длины указывает длину константы 25 байт. В константе записано меньше

25 знаков, поэтому справа после последнего знака А появятся 16 пробелов. В константе, определяемой оператором с именем NAME2, присутствуют 2 знака &&. В константе, построенной транслятором, этим двум знакам будет соответствовать только один знак &, т. е. будет создана константа длиной 9 байт, несмотря на то, что в операнде оператора DC записано 10 знаков. Пробел в знаковой константе внутри апострофов, заключающих константу, воспринимается как знак константы.

По оператору с именем NAME3 в памяти будет создана константа ABCDABCD. Явная длина, указываемая в этом операторе модификатором длины, равна 4, в машинную программу попадут только первые 4 знака константы из поля операндов, остальные знаки справа отбрасываются. Кратность, равная 2, в этом же операторе указывает, что константа должна повториться в машинной программе 2 раза. В операторе с именем NAME4 приведен пример использования в константе апострофа как знака константы. Если бы в константе был записан один апостроф, то это воспринималось бы транслятором, как конец константы.

Шестнадцатеричная константа. Информация, находящаяся в памяти ЕС ЭВМ, наглядно чаще всего записывается не в двоичном коде, а в шестнадцатеричном. Поэтому, если известно содержание памяти, можно определить эту информацию в программе с помощью шестнадцатеричной константы (тип X) в операторе DC.

При записи шестнадцатеричной константы можно использовать только шестнадцатеричные цифры: 0—9 и A, B, C, D, E, F.

В одном операнде оператора DC может быть указана одна шестнадцатеричная константа. Записываемая константа заключается в апострофы. Каждые две шестнадцатеричные цифры транслируются в один байт. Если число цифр, записанных в константе, нечетное, то самый левый байт созданной константы в левых четырех битах будет содержать нули, а в четырех правых — самую первую цифру. Выравнивание для шестнадцатеричной константы не производится.

Максимальная длина шестнадцатеричной константы (явная или неявная длина) равна 256 байт. Если модификатор длины не указан, то длина константы (неявная) равна половине количества шестнадцатеричных цифр, записанных в константе (предполагается, что шестнадцатеричный ноль добавлен к нечетному числу цифр). Если модификатор длины указан, то результат трансляции определяется следующим образом:

если число байт, необходимых для представления константы, превышает указанную длину, то самые левые цифры константы отбрасываются;

если число байт, необходимых для представления константы, меньше указанной длины, то у константы слева добавляется необходимое количество нулей.

Ниже приведены примеры записи операторов DC для определения шестнадцатеричных констант.

Название	Операция	Операнды
NAME	DC	XL2'F1F2F3F4'
NAME1	DC	XL4'FF'
NAME2	DC	3X'FF'
NAME3	DC	X'F1F2FM'
NAME4	DC	X'123'

По оператору с именем NAME в машинной программе будет построена константа длиной 2 байта, как это указано модификатором длины. В этих байтах будут находиться последние четыре шестнадцатеричные цифры: F3F4, остальные левые цифры будут отброшены. Следующий оператор с именем NAME1 создаст 4 байта, из которых первые три будут нулевыми (добавлены по требованию модификатора длины), а последний будет содержать X'FF'. По оператору с именем NAME2 будут построены три байта, значение каждого из которых равно X'FF'. Оператор с именем NAME3 неправильный, так как в константе записан символ M, не принадлежащий к шестнадцатеричным цифрам. В операторе с именем NAME4 записано нечетное число шестнадцатеричных цифр (три), в этом случае слева будет добавлен один нуль.

Двоичная константа. Программист может определить оператором DC информацию прямо в таком виде, как она представлена в памяти машины: в двоичном коде. Для этого предназначена двоичная константа (тип B). Это удобно для задания различных масок, значений переключателей и другой информации.

Двоичная константа записывается как последовательность нулей и единиц, заключенная в апострофы. Если количество нулей и единиц в записанной последовательности не кратно восьми, то транслятор слева добавляет недостающее (до кратности 8) число нулей. В одном операнде оператора DC может быть определена только одна двоичная константа. Выравнивание для двоичной константы не производится.

Максимальная длина двоичной константы (явная или неявная) равна 256 байт. Если не указан модификатор длины, то длина двоичной константы равна числу байт, занимаемых константой в памяти. Если в операторе DC задан модификатор длины (указана явная длина константы), то результат трансляции определяется следующим образом:

если число байт, которое требуется для представления двоичной константы, превышает указанную длину, то самые левые цифры константы отбрасываются;

если число байт, которое требуется для представления двоичной константы, меньше указанной длины, то слева добавляется необходимое число нулевых бит.

Ниже приведены примеры записи двоичных констант (с. 149).

Константа, построенная по оператору с именем NAME1, будет занимать один байт. Байт, построенный по оператору с именем

Название	Операция	Операнды
NAME1	DC	B'11110000'
NAME2	DC	B'101'
NAME3	DC	BL2'1111'
NAME4	DC	BL1'111110000'
NAME5	DC	B'10210001'

NAME2, слева будет содержать 5 нулевых бит. По оператору с именем NAME3 будет построена константа длиной два байта, в которой первый байт и четыре первых разряда второго байта будут нулевыми, а четыре последних разряда второго байта будут содержать единицы. В операторе с именем NAME4 указана явная длина, равная 1, но в константе записано нулей в единиц больше, чем необходимо для одного байта. Три левые единицы будут отброшены. Оператор с именем NAME5 неправильный, так как в константе записана цифра 2, что для констант типа B не допускается.

Десятичные константы. Система команд ЕС ЭВМ позволяет выполнять действия над десятичными числами, которые должны быть представлены в памяти в одном из двух форматов: в упакованном десятичном формате или в десятичном формате с зоной. Эти числа можно определить оператором DC: упакованная десятичная константа (тип P) определяет упакованные десятичные числа, распакованная десятичная константа (тип Z) определяет десятичные числа с зоной. В обоих случаях в поле операндов оператора DC записывается десятичная константа, заключенная в апострофы. Исходя из указанного типа (P или Z) транслятор представит в машинной программе десятичную константу в виде десятичного упакованного числа или десятичного числа с зоной.

Десятичная константа записывается как десятичное число со знаком или без знака. Если знак опущен, то предполагается знак плюс. При записи десятичной константы программист может внутри константы в любом месте записать точку (чтобы отметить для себя положение десятичной точки). Записанная точка не влияет на результат трансляции, при трансляции она пропускается. В одном операнде оператора DC может быть определено несколько десятичных констант. Вывравнивание для десятичных констант не производится.

Если оператор DC определяет упакованную десятичную константу (тип P), то каждая пара из записанных десятичных цифр транслируется в один байт. Самая правая цифра помещается со знаком числа в самый правый байт. Каждая десятичная цифра представляется с помощью четырех двоичных разрядов так же, как шестнадцатеричные цифры 0—9. Если в упакованной десятичной константе указано четное число цифр, то в этом случае в самом левом байте четыре левых бита будут нулевыми, а четыре правых бита будут содержать первую цифру, потому что самая

правая цифра числа располагается в одном байте со знаком числа.

Если оператор DC определяет распакованную десятичную константу (тип Z), то каждая десятичная цифра константы транслируется в один байт. Первые четыре бита каждого байта, кроме самого правого, будут содержать единицы (зону), а следующие четыре — цифру. Самый правый байт будет содержать знак и самую правую цифру.

Для обеих десятичных констант, упакованной десятичной и распакованной десятичной, знак плюс транслируется в шестнадцатеричную цифру C, а знак минус — в цифру D.

Максимальная длина десятичной константы (явная или неявная) равна 16 байт. Если в операторе DC, определяющем десятичную константу, не указан модификатор длины, неявная длина любой десятичной константы равна числу байт, занимаемых константой, с учетом формата, знака и добавления нулевых битов для упакованных десятичных констант. Если указан модификатор длины, то результат трансляции определяется по следующим правилам:

если для представления константы требуется меньше байт, чем указывает модификатор длины, то в константе слева добавляется необходимое количество нулей;

если для представления константы требуется больше байт, чем указывает модификатор длины, то необходимое число цифр в константе усекается слева.

Ниже приведены примеры записи десятичных констант.

Название	Операция	Операнды
NAME	DC	P'12345'
NAME1	DC	PL2'+5'
NAME2	DC	PL1'—75·34'
NAME3	DC	PL2'2,+2·54,—459736'
NAME4	DC	Z'12345'
NAME5	DC	ZL2'+1·Ø45'
NAME6	DC	ZL4'—89'
NAME7	DC	P'125A67'

По оператору с именем NAME будет построено упакованное десятичное число длиной 3 байта, которое в шестнадцатеричном виде представляется как X'12345C'. По оператору с именем NAME1 построится упакованное десятичное число длиной два байта, как это указывает модификатор длины. Цифра 5 и знак плюс составляют правый байт, а слева будет добавлен один нулевой байт. Десятичное число, построенное по оператору с именем NAME2, будет представлять собой один байт, содержащий цифру 4 и знак минус. Остальные цифры, записанные в константе (753), будут отброшены исходя из указания модификатора длины. Точка, записанная в константе, будет пропущена. В операторе с име-

нем NAME3 определяются три константы. Модификатор длины указывает длину, равную двум, для всех трех констант. Для того чтобы в машинной программе каждая константа занимала 2 байта, в первой константе добавится нулевой байт, а в третьей константе будут отброшены левые цифры (459).

Оператор с именем NAME4 определяет распакованную десятичную константу, по этому оператору будет построено десятичное число с зоной длиной 5 байт. Последний пятый байт будет содержать цифру 5 и знак плюс. По оператору с именем NAME5 построится десятичное число с зоной длиной два байта, как этого требует модификатор длины. Левые цифры (1 и 0) будут отброшены. Точка, записанная в константе, будет пропущена. Знак плюс будет содержаться в самом левом байте вместе с цифрой 5. В константе, построенной по оператору с именем NAME6, слева будут добавлены два байта, содержащие десятичную цифру нуль с зоной. Оператор с именем NAME7 неправильный, так как в константе записана не десятичная цифра.

Константы с фиксированной точкой. Команды с фиксированной точкой ЕС ЭВМ выполняют действия над данными, имеющими определенный формат. Эти данные — числа с фиксированной точкой. Формат чисел с фиксированной точкой рассматривался в 1.2.

Определять числа с фиксированной точкой в программе в таком виде, как они представляются в памяти машины, довольно трудно. Проще определять числа с фиксированной точкой, указывая в операнде оператора DC тип F или H. В этом случае в операнде записываются константы в обычной десятичной форме, а представление этих констант в форме чисел с фиксированной точкой выполняет транслятор. Указывая в операнде оператора DC тип H, можно определить короткое число с фиксированной точкой (2 байта), а указав тип F — длинное число с фиксированной точкой (4 байта). В одном операнде оператора DC можно записывать несколько констант типа F или H.

Константа или константы, записываемые в операнде оператора, должны заключаться в апострофы. Каждая константа в операнде записывается как десятичное число со знаком или без знака, за которым может следовать десятичный порядок. Если число записано без знака, то предполагается знак плюс. Число может быть целым, дробным или смешанным (числом с целой и дробной частью). Дробная часть при записи числа отделяется точкой. Точка может быть записана до, после или внутри числа или может быть опущена. Например, следующие три числа будут равнозначны: 423, 423., 423.00. Присутствие порядка при записи числа не обязательно. Если необходимо указать порядок, то он записывается непосредственно после десятичного числа как E_n , где n — десятичный самоопределенный терм со знаком или без знака. Если знак порядка опущен, подразумевается знак плюс. Если порядок указан, то число перед его преобразованием в машинный формат умножается на 10 в степени, равной порядку.

В операторе DC, определяющем числа с фиксированной точкой, может быть использован модификатор порядка. Модификатор порядка выполняет те же функции, что и порядок, записанный непосредственно в константе. Но модификатор порядка относится ко всем константам, записанным в операнде, а порядок, записанный непосредственно в константе, относится только к этой константе. Модификатор порядка должен находиться в интервале от -85 до $+75$. Порядок, записываемый в числе, может выходить из этого интервала, но только в том случае, если сумма порядка числа и модификатора порядка не выходит из этого интервала. Правила записи модификатора порядка см. в 3.6.1.

Если в операнде оператора DC не указан модификатор длины, то неявная длина константы типа F предполагается равной четырем байтам, типа H — двум байтам. В этом случае константа типа F выравнивается на границу слова, типа H — на границу полу-слова. Модификатором длины для обоих типов констант может быть указана любая длина от одного до восьми байт включительно. Если длина константы задана явно, то выравнивание не производится.

Для константы с фиксированной точкой может указываться модификатор масштаба. Правила записи модификатора масштаба см. в 3.6.1. Если для константы с фиксированной точкой указан модификатор масштаба, то транслятор после перевода константы в двоичное представление (отдельно переводится целая и дробная часть) умножает ее на 2 в степени, равной модификатору масштаба. Значение модификатора масштаба может задаваться в интервале от -187 до $+346$. Фактически масштабирование вызывает перемещение двоичной точки, отделяющей целую и дробную часть. Поэтому модификатор масштаба для констант с фиксированной точкой определяет:

число двоичных позиций дробной части, которые присоединяются к целой части двоичного числа, если масштаб положительный;

число двоичных позиций, которые должны быть исключены из целой части двоичного числа, если масштаб отрицательный.

Таким образом, используя масштабирование, программист может определять числа с фиксированной точкой с учетом их дробной части. Если не указан модификатор масштаба, транслятор после перевода константы в двоичное представление отбрасывает дробную часть. Но, присоединяя дробную часть к числам с фиксированной точкой, программист при выполнении действий над такими числами должен сам следить за положением точки, разделяющей дробную и целую части числа.

После перевода константы в двоичное представление и масштабирования или (если отсутствует масштабирование) сразу после перевода двоичное число помещается в соответствующую область памяти согласно явной или неявной длине, причем производится округление результата в зависимости от величины отбрасы-

ваемой дробной части. Полученное число отличается от точного не более чем на единицу в последнем бите.

Если размер переведенного числа превышает необходимую длину, явную или неявную, то теряется знак числа и самые левые биты числа. Нулевые биты будут добавлены слева, если длина двоичного числа меньше необходимой длины. Отрицательные числа помещаются в память в дополнительном коде.

Ниже приведены примеры записи констант с фиксированной точкой.

Название	Операция	Операнды
NAME	DC	3F'125'
NAME1	DC	HS4'—25.45'
NAME2	DC	FL8E2'1Ø.257,1ØØ,1ØE—3'

По оператору с именем NAME будут построены три числа с фиксированной точкой длиной в слово, каждое из которых в памяти будет представлено как X'0000007D'. Кратность, равная 3, вызовет построение этого числа три раза. Оператор с именем NAME1 определяет отрицательное число с фиксированной точкой длиной два байта. В памяти это число будет представлено в дополнительном коде как X'FE69'. Из двоичной дробной части к числу будут присоединены 4 бита, как этого требует модификатор масштаба.

Оператор с именем NAME2 определяет три числа с фиксированной точкой, каждое число будет занимать в памяти 8 байт, как этого требует модификатор длины. Модификатор порядка вызывает умножение перед переводом всех констант, записанных в операнде, на 100 (10^2). На самом деле оператором определяются константы: 1025.7, 10000, 1000E—3. Порядок, записанный в третьей константе, относится только к этой константе, т. е. оператор определяет следующие константы с фиксированной точкой: 1025.7, 10000, 1. Двоичная дробная часть у первого числа будет отброшена, так как модификатор масштаба отсутствует.

Константы с плавающей точкой. Команды с плавающей точкой выполняют действия над числами с плавающей точкой длиной 8 байт (длинными) и 4 байта (короткими). Формат чисел с плавающей точкой рассматривается в 1.2. Оператор DC, в котором указан тип E или D в операнде команды, позволяет определить соответственно короткие и длинные числа с плавающей точкой.

Правила записи констант в поле операндов оператора DC при определении чисел с плавающей точкой полностью совпадают с правилами записи констант при определении чисел с фиксированной точкой. В одном операнде оператора DC может быть определено несколько констант типа E или D.

При определении чисел с плавающей точкой может указывать-

ся модификатор порядка. Диапазон модификатора порядка и порядка константы такой же, как при определении чисел с фиксированной точкой. Умножение десятичного числа, записанного в операторе DC на 10 в степени, указываемой модификатором порядка или порядком, записанным в самой константе, выполняется до преобразования констант в машинный формат. Если в операнде оператора DC не указан модификатор длины, то неявная длина констант типа E предполагается равной 4 байтам, типа D — 8 байтам. В этом случае константы типа E выравниваются на границу слова, типа D — на границу двойного слова. Модификатором длины для обоих типов констант с плавающей точкой может быть указана любая длина от одного до восьми байт включительно. Выравнивание при задании явной длины не производится.

Преобразование в машинный формат констант с плавающей точкой типа E и D выполняется одинаково. Константы этих типов отличаются только длиной мантиссы. Преобразование выполняется следующим образом. Каждая константа с плавающей точкой преобразуется в шестнадцатеричную нормализованную дробь. В операнде оператора DC может быть указан модификатор масштаба. В этом случае после перевода выполняется масштабирование.

Модификатор масштаба для констант с плавающей точкой может указываться только положительным, и его значение должно находиться в диапазоне от 1 до 14. Он указывает число шестнадцатеричных позиций, на которые должна быть сдвинута шестнадцатеричная дробь вправо (каждая шестнадцатеричная позиция состоит из четырех разрядов), т. е. масштаб указывает число шестнадцатеричных нулей, которые должны появиться слева у нормализованной шестнадцатеричной дроби. Таким образом, масштабирование, указанное для константы с плавающей точкой, порождает ненормализованное число. При масштабировании порядок константы корректируется, чтобы сохранить правильную величину константы.

После масштабирования или, если отсутствует масштабирование, сразу после перевода мантиссы формируется характеристика, и константа в формате числа с плавающей точкой помещается в соответствующую область памяти.

Если длина преобразованного числа превышает указанную или неявную длину, то усекаются самые правые биты мантиссы. При этом производится округление мантиссы с учетом величины отбрасываемой части мантиссы. Окончательное число не будет отличаться от точного значения больше, чем на единицу в младшем бите.

Ниже приведены примеры записи констант с плавающей точкой, в комментариях приведено шестнадцатеричное представление каждой константы в памяти.

Первый байт в шестнадцатеричном представлении — это характеристика. У отрицательных чисел нулевой бит этого байта — единица.

Название	Операция	Операнды
	DC	E'-4' C110000000
	DC	E'+9' 419000000
	DC	E'-1.5' C11800000
	DC	E'1.125' 41120000
	DC	E'-0.01' BF28F5C3
	DC	E'0.0001' 3D68DB8C
	DC	D'1' 411000000000000000
	DC	D'12345678912345' 4BB3A73CE5B59000
	DC	D'-1.65789516' C11A86BD134658D5
	DC	D'0.5' 408000000000000000

Ниже приведены различные способы записи одной и той же константы с плавающей точкой. В шестнадцатеричном представлении эта константа выглядит как X'422E6A3D'.

Название	Операция	Операнды
	DC	E'46.415'
	DC	E'46415E-3'
	DC	EE2'.46415'
	DC	EE+3'0.46415E-1'

Следующий оператор DC определяет несколько констант с плавающей точкой длиной 8 байт:

Название	Операция	Операнды
	DC	DS2E+2'+46,-3.729,+473E-1'

Модификатор порядка, равный 2, относится ко всем константам; порядок, равный -1, относится только к последней константе. Таким образом, будут переводиться в машинный формат следующие константы: +4600, -372,9, +4730. Модификатор масштаба, равный 2, относится к каждой константе и будет вызывать появление в начале мантиисы двух шестнадцатеричных нулей.

Модификатор длины в битах. Транслятор Ассемблера F допускает задание модификатора длины в битах. В этом случае модификатор длины указывает число бит, которое требуется для размещения константы в памяти. Задание длины в битах допускается только для констант типа C, X, B, P, Z, H, F, E и D.

Модификатор длины в битах записывается как L.n, где n — десятичный самоопределенный терм без знака или положительное абсолютное выражение, заключенное в скобки. L и n должны раз-

деляться точкой. Значение n указывает число бит, которое требуется для размещения константы в памяти. Модификатор длины в битах указывает, что константа будет занимать целое число байт плюс некоторое число бит. Например, $L, 20$ означает, что длина константы 2 байта и 4 бита.

Модификатор длины в битах не должен превышать максимально допустимого значения длины в байтах для каждого типа констант. Каждая константа перед помещением ее в память доподняется или усекается согласно заданной длине в битах (слева или справа в соответствии с типом константы). Первая или единственная константа, определяемая каждым оператором DC с модификатором длины в битах, всегда начинается с нулевого бита первого занимаемого байта. Если построенная константа закончится не на границе байта, а за ней непосредственно в данном операторе не следует другая константа с длиной в битах, то остаток последнего байта заполняется нулевыми битами. В счетчике адреса в этом случае устанавливается значение адреса следующего байта. Если для следующей константы, определяемой в этом операторе, указан модификатор длины в битах, то она располагается с очередного свободного бита. Если в операторе DC с модификатором длины в битах указана кратность, то дополнение нулевыми битами выполняется только один раз, а именно, в конце поля, занятого последней дублируемой константой.

3.6.3. Адресные константы

Адресная константа — это адрес основной памяти, используемый в программе как константа. Если необходимо выполнить действия над адресом, можно определить его как константу с помощью команды Ассемблера DC, а затем выполнять над ним нужные операции, как над числом с фиксированной точкой.

Примером использования адресных констант является занесение адреса в регистр базы для адресации основной памяти: адрес памяти определяется как адресная константа, а затем эта константа загружается в регистр базы.

Адресные константы, в отличие от констант других типов, при записи операнда оператора DC заключаются в скобки. В одном операнде можно определять несколько адресных констант, в этом случае они отделяются друг от друга запятыми, а вся последовательность адресных констант заключается в круглые скобки.

Оператором DC можно определять следующие адресные константы: действительные (тип A и Y), явные (тип S) и внешние (тип V).

Действительные адресные константы. Действительные адреса памяти могут использоваться в программе в виде констант. Такие константы называются действительными адресными константами. Они могут быть типа A и типа Y.

Действительная адресная константа типа A задается абсолютным, простым переместимым или составным переместимым выра-

жением (типы выражений см. в 2.1.7). Значение выражения не должно превышать $2^{31}-1$. После того как транслятор вычислит значение этого выражения, он усекает значение слева до явной или неявной длины и размещает его в самых правых битах области, отведенной под константу.

Неявная длина константы типа А равна четырем байтам. В случае неявной длины для констант типа А выполняется выравнивание на границу слова. Если длина указана, выравнивание не производится. Значение модификатора длины зависит от типа выражения, используемого для определения действительной адресной константы: для абсолютного выражения может быть указана длина от 1 до 4 байт, для простого переместимого или составного переместимого выражения может быть указана длина только 3 или 4 байта.

Если адресная константа содержит значение счетчика адреса, значение счетчика адреса равно адресу первого байта области, занимаемой константой в памяти. Если в операнде оператора счетчик адреса используется в нескольких адресных константах, то значение счетчика адреса изменяется от константы к константе. Аналогично, если в операторе DC указана кратность, а в этом операторе определяется адресная константа, использующая счетчик адреса, то константа копируется с изменением счетчика адреса.

Ниже приведены примеры адресных констант типа А.

Название	Операция	Операнды
ACON	DC	A(NAME)
ACON1	DC	A(1ØØ)
ACON2	DC	A(END-5ØØ)
ACON3	DC	A(*+4Ø96)

По оператору с именем ACON будет построена константа длиной 4 байта, расположенная на границе слова, значение ее будет равно значению символического имени NAME. Если NAME — переместимое символическое имя, то это значит, что адресная константа определена простым переместимым выражением. Следующая адресная константа с именем ACON1 определена абсолютным выражением. Константы с именами ACON2 и ACON3 определяются простыми переместимыми выражениями (учитывая, что END — переместимое символическое имя). Каждая из приведенных констант занимает четыре байта памяти.

Можно определить адресную константу, занимающую два байта памяти. Для этого в операторе DC необходимо указать тип Y. Адресная константа типа Y определяется так же, как и константа типа А, только окончательное значение выражения, определяющего адресную константу, не должно превышать $2^{15}-1$. Вычисленное значение выражения усекается слева до явной или неяв-

ной длины и помещается в самые правые биты области, отведенной для константы. Неявная длина константы типа Y равна двум байтам. В случае неявной длины для констант типа Y выполняется выравнивание на границу полуслова. Если длина не указана, выравнивание не производится. Значение модификатора длины для константы типа Y, определяемой переместимым выражением (простым или составным), равно двум байтам, а для констант, определяемых абсолютным выражением, равно одному или двум байтам.

Счетчик адреса в адресных константах типа Y обрабатывается точно так же, как и в константах типа A. Приводимые ранее константы можно было определить и с помощью констант типа Y, как это показано ниже.

Название	Операция	Операнды
	DC	Y(NAME)
	DC	Y(1ØØ)
	DC	Y(END-5ØØ)
	DC	Y(*+4Ø96)

В этом случае по операторам DC будут построены константы длиной 2 байта, при этом они располагаются на границе полуслова.

Кроме приводимых ранее примеров использования адресных констант для загрузки регистров базы, можно рассмотреть использование адресных констант при организации переходов. Например, следующая последовательность команд вызовет переход на команду с именем NAME:

Название	Операция	Операнды
...	L ...	3,ADCON
ADCON	BR	3
	DC	A(NAME)
NAME	LA ...	1,2 ...
...

Значения действительных адресных констант, определяемых переместимыми выражениями, устанавливаются транслятором относительно условного начального адреса программы. Окончательные значения переместимым выражениям устанавливаются Редактором исходя из адреса загрузки программы.

Явная адресная константа. В программах можно использовать константы в виде явных адресов памяти (в форме «база—смещение»). Такие константы называются явными адресными

константами, или константами типа S. Явная адресная константа может быть определена двумя способами: как одно абсолютное или простое переместимое выражение, например DC S(NAME); как два абсолютных выражения, первое из которых представляет смещение, а второе — регистр базы. Второе выражение заключается в скобки, например DC S(400(13)).

В первом случае выражение определяет неявный адрес, во втором случае два выражения определяют явный адрес. Выражение, определяющее неявный адрес, будет преобразовано транслятором к форме «база — смещение», т. е. для неявного адреса выбирается доступный регистр базы и вычисляется смещение.

Явная адресная константа всегда занимает два байта. Четыре самых левых бита константы определяют регистр базы, остальные 12 бит — смещение. Если длина константы задается явно, то она может указываться равной только двум байтам, выравнивание в этом случае не производится. Если длина задается неявно, то производится выравнивание на границу полуслова.

Явная адресная константа может использоваться при формировании команды на машинном языке. В этом случае адрес для формируемой команды может быть подготовлен в форме «база — смещение» с помощью явной адресной константы.

Внешняя адресная константа. Программист может зарезервировать в своей программе область памяти для значения простого символического имени из другой программы, т. е. имени, используемого для связи с другой программой и для обращения к внешним данным. Это можно сделать, определив в операторе DC внешнюю адресную константу (тип V).

Константа типа V задается одним переместимым символическим именем. Символическое имя определяется в другой программе, поэтому транслятор не может установить значение этого имени, а только резервирует для него место. Значение константы после трансляции будет равно нулю, окончательное значение устанавливается Редактором.

Неявная длина константы типа V равна 4 байтам. Модификатором длины для константы типа V можно указывать длину только 3 или 4 байта. Если модификатор длины не указывается, то производится выравнивание на границу слова, если указывается, то выравнивание не производится.

По оператору DC, приведенному ниже, будут резервироваться 12 байт, так как оператор определяет три внешних константы.

Название	Операция	Операнды
	DC	V(NAME,CONS,VCON)

Внешние адресные константы используются для связи между отдельно транслируемыми программами (см. 3.11).

3.7. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ЛИТЕРАЛОВ

В машинных командах в качестве операндов можно использовать литералы. Литерал — это константа, необходимая машинной команде для ее выполнения и записанная в команде на месте адреса.

Формат записи литерала почти идентичен формату записи константы в операнде команды Ассемблера DC. Все правила записи операнда оператора DC относятся и к написанию литералов. Имеются только следующие различия:

литералу предшествует знак =;

в качестве кратности и модификатора длины может использоваться только десятичный самоопределенный терм без знака;

в качестве модификатора масштаба и модификатора порядка может использоваться только десятичный самоопределенный терм без знака или со знаком;

кратность не может быть равна нулю;

в литерале не может быть определена константа типа S.

Литералы могут использоваться в тех машинных командах, в которых адреса памяти являются операндами. При использовании литералов необходимо соблюдать следующие правила:

в команде может быть записан только один литерал;

литерал должен записываться на месте того операнда команды, содержимое которого не изменяется при выполнении этой команды;

литерал должен быть единственным термом в операнде;

литерал не может использоваться в командах сдвига и ввода—вывода.

Рассмотрим пример использования литерала. Допустим, необходимо загрузить в регистр число с фиксированной точкой, равное 100. Не используя литерал, это можно выполнить, записав следующие операторы:

Название	Операция	Операнды
...	L ...	3,CONST
CONST	DC ...	F'100'

С помощью литерала это можно сделать одним оператором:

Название	Операция	Операнды
...	L ...	3,=F'100'
...

При программировании удобнее использовать литералы, потому что в этом случае сразу видны данные, над которыми производятся действия, а размещение константы в памяти выполняется транслятором.

Рассмотрим следующий пример использования литерала:

Название	Операция	Операнды
NAME ...	L ...	10, = A(*+4095) ...

В приведенной команде литералом является адресная константа, в которой используется счетчик адреса. В этом случае значением счетчика адреса является адрес команды (значение имени NAME), а не адрес литерала, т. е. области, где будет размещена адресная константа.

Необходимо отличать литералы от непосредственных операндов в команде формата SI. Непосредственные операнды сами транслируются в команду, а при использовании литерала в команде появляется адрес константы.

Допустим, в программе присутствуют следующие команды:

Название	Операция	Операнды
...	MVI MVC MVC	OBL1, X'FF' OBL2(1), = X'FF' OBL(16), = 4C'ABCD'
OBL1 OBL2 OBL	DS DS DC	C C CL16

В команде MVI записан непосредственный операнд X'FF', который будет занимать разряды 8—15 в построенной машинной команде. В команде MVC используется литерал =X'FF'. В этом случае после трансляции эта константа будет расположена в памяти, а в команде построен ее адрес. Во второй команде MVC приведен литерал, в котором используется кратность. Как и в случае определения константы с помощью оператора DC, константа ABCD появится в памяти 4 раза. В команду MVC после трансляции будет подставлен адрес первого байта первой константы.

Литералы при обработке программы транслятором Ассемблера собираются и размещаются в специальной области памяти, называемой областью литералов. При желании программист может управлять размещением области литералов в своей программе. Для этого предназначена команда Ассемблера LTORG.

Команда Ассемблера LTORG определяет начало области литералов. В эту область помещаются литералы, используемые в тех командах программы, которые расположены, начиная от предыдущего оператора LTORG (или от начала программы, если оператор LTORG еще не использовался), до данного оператора LTORG.

Началом области литералов всегда является адрес первого двойного слова, следующего за оператором LTORG. Если перед оператором LTORG литералов в программе нет, то по оператору LTORG выполняется только выравнивание на границу двойного слова. Байты, пропущенные при выравнивании на границу, нулями не заполняются.

Оператор LTORG имеет следующий формат:

Название	Операция	Операнды
Любое символическое имя или пробел	LTORG	Не используется

Символическое имя в поле названия представляет адрес первого байта области литералов.

Программист в программе может создавать несколько областей литералов, т. е. записывать в программе несколько операторов LTORG. Например, в программе имеется следующая последовательность команд:

Название	Операция	Операнды
...	MVC	A, =F'1'
OBL	AD	2, =D'2'
	LTORG	
	IC	5, =X'FF'
OBL1	L	6, =F'100'
	LTORG	
...

В этом случае будут образованы две области литералов: первая — по оператору LTORG с именем OBL, вторая — по оператору LTORG с именем OBL1. Литералы =F'1', =D'2' будут помещены в первую область литералов (OBL), литералы =X'FF', =F'100' — во вторую.

Литералы, у которых запись одинаковая, называются дублирующими. Только первый из дублирующих литералов, относящихся к одной области литералов, помещается в эту область. Если литерал является адресной константой, содержащей счетчик адреса, то в область литералов помещается каждый литерал, даже если он дублирующий.

Рассмотрим для примера следующую последовательность команд:

Название	Операция	Операнды
...	L ...	3, = X'F0'
	L	4, = C'0'
	L	5, = X'FFFF'
OBL	LTORG	
	L	10, = A(*+4)
	L	11, = A(*+4)
	LH	5, = X'FFFF'
NAME	LH	12, = X'FF10'
NAME1	LH	13, = X'FF10'
OBL1	LTORG	
...

Литералы $=X'F0'$ и $=C'0'$ различаются по записи, поэтому оба будут помещены в область литералов, определяемую оператором LTORG с именем OBL, хотя в машинной программе они представляют одинаковые константы. В область литералов с именем OBL1 помещаются оба литерала $=A(*+4)$, так как эти литералы определяют адресные константы, использующие счетчик адреса. Они используются в разных командах и поэтому, несмотря на полную идентичность записи, будут иметь различные значения. Литерал $=X'FFFF'$ будет помещен в каждую из двух областей литералов. Литералы в командах с именами NAME и NAME1 ($=X'FF10'$) полностью идентичны по записи и относятся к одной области литералов OBL1. Они являются дублирующими. В область литералов OBL1 помещается только первый из них.

Таким образом, с помощью операторов LTORG можно управлять расположением литералов в памяти. Однако последовательность, в которой располагаются литералы внутри области литералов, определяется транслятором. Область литералов состоит из четырех разделов, в которых литералы расположены согласно их длине. Первый раздел содержит все литералы с длиной, кратной 8, второй раздел — с длиной, кратной 4, третий раздел — с длиной, кратной 2, четвертый — все литералы с нечетной длиной.

В программе могут не использоваться операторы LTORG или могут присутствовать литералы после последнего оператора LTORG. Тогда область литералов размещается в конце первой программной секции. В этом случае необходимо, чтобы первая программная секция была адресуема, т. е. чтобы существовал регистр базы, которым можно базировать адреса литералов. Регистр базы не должен изменяться при работе других секций, так как в этих секциях тоже могут использоваться литералы, расположенные в первой программной секции. Такой случай будет рассматриваться в 3.11.

Допустим, в приведенной ниже программе отсутствуют операторы LTORG:

Название	Операция	Операнды
BEGIN	BALR	15,0
...	USING	*,15
...	LA	1,=X'FF'
...	L	2,=C'ABCD'
	END	BEGIN

В программе используются литералы. После трансляции литералы будут расположены в конце программы (программа состоит из одной программной секции), а в распечатке они будут присутствовать после оператора END, который является последним оператором исходной программы. В приведенной программе необходимо проследить, чтобы для представления адресов литералов был определен доступный регистр базы. В программе регистром базы является регистр 15. Базовый адрес равен адресу BEGIN+2. Если литералы будут находиться от адреса BEGIN на расстоянии больше 4097 байт, то адреса литералов невозможно базировать регистром 15, так как смещение будет получено больше 4095. В этом случае нужно в программе определить еще один регистр базы, доступный и для адресов литералов.

3.8. РЕЗЕРВИРОВАНИЕ И ОПИСАНИЕ ОБЛАСТЕЙ ПАМЯТИ

В любой программе необходимо иметь рабочие области памяти. Начальное содержимое этих областей определять нет необходимости, нужно только зарезервировать области памяти определенной длины. Эту задачу можно выполнить с помощью команды Ассемблера DS (ОПРЕДЕЛИТЬ ПАМЯТЬ).

3.8.1. Оператор определения области памяти

Оператор DS позволяет определять области памяти и присваивать этим областям символические имена. По этому оператору транслятор ничего не строит в памяти, а только резервирует указанное оператором количество байт памяти, т. е. увеличивает на это количество значение счетчика адреса.

Оператор DS имеет следующий формат:

Название	Операция	Операнды
Любое символическое имя или пробел	DS	Один или несколько операндов, разделенных запятыми

Формат операнда оператора DS идентичен формату операнда оператора DC. В нем присутствуют те же элементы и записываются в такой же последовательности, как и в операнде оператора (кратность, тип, модификаторы, константы). Однако имеются два различия в задании операнда:

указание константы в операнде оператора DS не обязательно; максимальная длина, которая может указываться для знаковых (тип C) и шестнадцатеричных (тип X) констант, равна 65535 байтам, а не 256.

Если в операнде оператора DS записаны константы, то они в случае отсутствия модификатора длины используются транслятором для определения длины области, которую нужно зарезервировать. Построение констант в памяти, несмотря на присутствие констант в операнде, по оператору DS не выполняется.

Чтобы в программе можно было обращаться к области памяти, резервируемой оператором DS, в поле названия этого оператора можно записывать символическое имя. Значение этого символического имени — это адрес самого левого байта резервируемой области памяти. Характеристика длины имени равна длине первой (или единственной) области, резервируемой данным оператором DS. Кратность, указанная в операнде оператора DS, не влияет на характеристику длины. Если в операторе DS не указан модификатор длины, то выполняется выравнивание таким же образом, как и для оператора DC (в зависимости от типа константы). Если модификатор длины присутствует в операнде оператора DS, выравнивание не производится. Байты, пропущенные при помещении области, определяемой оператором DS, на границу, ничем не заполняются.

Рассмотрим примеры резервирования областей памяти.

Допустим, программа должна ввести запись с внешнего устройства в некоторую область памяти, следует обработать эту запись и результат обработки разместить в области вывода. Длина области ввода равна 80 байтам, длина области вывода — 400 байтам. Кроме того, при обработке используется рабочая область длиной 8 байт. В программе, выполняющей поставленную задачу, необходимо зарезервировать три области памяти: для ввода, для обработки, для вывода. Ниже приведены операторы, которые можно было бы использовать для этой цели.

Название	Операция	Операнды
...	LM	2,3,BBOD
BBOD	DS	80C
RAB	DS	D
BIBOD	DS	XL400
...

Область для ввода записи резервируется оператором DS с именем BBOD. Это имя используется в команде LM для обращения к информации в этой области (предположим, ввод записи был выполнен до команды LM). В операторе DS, определяющем область ввода, указаны только кратность и тип. В этом случае транслятор резервирует область исходя из неявной длины, которая определяется типом константы, записанной в операнде. Если указан тип C, X, B, P, Z, неявная длина принимается равной 1, тип D — равной 8, тип A, V, F, E — равной 4, тип H, S — равной 2. Соответственно будет определяться и характеристика длины имени, именующего оператор DS, в поле операндов которого не указаны модификатор длины и константа. В приведенном примере неявная длина резервируемой по оператору с именем BBOD области равна 1, так как в операнде оператора указан тип C. Но в операнде оператора указана кратность 80, поэтому по оператору BBOD зарезервируется 80 байт. Следует учесть, что при такой записи характеристика длины имени BBOD будет равна 1 (кратность не влияет на характеристику длины). Это необходимо учитывать при использовании имени BBOD в командах, в которых будет использоваться неявная длина операндов.

Значение счетчика адреса будет увеличено в результате обработки оператором DS с именем BBOD на 80. В программу на машинном языке в эти 80 байт транслятор не помещает никакой информации. Информация здесь может появиться только во время выполнения программы. Для резервирования 80 байт можно также записать оператор DS одним из следующих способов:

Название	Операция	Операнды
BBOD1	DS	CL8Ø
BBOD2	DS	1ØCL8
BBOD3	DS	XL8Ø
BBOD4	DS	8ØX
BBOD5	DS	1ØD

В первых трех случаях характеристика длины имени, записанного в поле названия, равна значению модификатора длины. В последних двух случаях характеристика длины определяется типами X и D, записанными в операнде, и равна 1 и 8.

Рассмотрим, как определяется рабочая область для обработки, которая по условию должна быть длиной 8 байт. Зарезервировать такую область можно с помощью оператора DS любого типа. Но для резервирования областей длиной до 8 байт выгодно использовать операторы DS типа F, D, E и H, потому что тогда можно одновременно сделать выравнивание на нужную границу полуслова, слова или двойного слова. В данном случае для резервирования области RAB используется оператор DS типа D. Значение символического имени RAB будет адресом первого байта

области. Восьмибайтовая область будет начинаться на границе двойного слова (адрес будет кратен восьми), так как в операнде нет модификатора длины, а в этом случае для типа D выполняется выравнивание на границу двойного слова. Для резервирования области длиной четыре байта, расположенной на границе слова, можно написать один из следующих операторов:

Название	Операция	Операнды
	DS DS	E F

Для определения области длиной 8 байт, но без выравнивания на границу двойного слова можно записать оператор DS любого из типов F, E, H, D с указанием модификатора длины (например, DS FL8 или DS HL8). Однако для резервирования областей длиной больше 8 байт с помощью типов F, E, H, D необходимо использовать кратность, так как максимальная допустимая длина для них — 8.

Для констант типа C и X максимальное значение модификатора длины равно 65535. Применение этих типов констант позволяет расширить диапазон длин резервируемых областей памяти. Область вывода, которую необходимо определить по условию поставленной ранее задачи, должна быть длиной 400 байт. Эта область определяется оператором DS с именем BIBOD. В этом операторе задан тип константы X и модификатор длины 400. Значение счетчика адреса увеличивается транслятором по этому оператору на 400. Выравнивание для типа X не производится.

В рассматриваемых примерах в поле операндов оператора DS константы отсутствовали, хотя они могут там записываться. Если в поле операндов оператора DS записаны константы, то длина резервируемой области по оператору DS в этом случае определяется следующим образом:

если в операнде указан модификатор длины, то присутствие констант не оказывает никакого действия. Длина резервируемой области определяется значением модификатора длины и кратностью;

если в операнде не указан модификатор длины, длина резервируемой области определяется длиной записанных в операнде констант и кратностью.

Например, по оператору DS C'ABCDEF' будет зарезервировано 6 байт памяти. В зарезервированной области транслятором никакой информации построено не будет, так как константа, записанная в операнде оператора DS, служит только для определения длины необходимой области.

3.8.2. Нулевая кратность

В операндах операторов DC и DS может указываться нулевая кратность.

Нулевая кратность вызывает почти полностью одинаковые действия при использовании ее в операторе DC и в операторе DS. Если указать в операнде оператора DC или DS нулевую кратность, то, если отсутствует модификатор длины, выполняется только выравнивание значения счетчика адреса на границу согласно типу, указанному в операнде:

- тип D — на границу двойного слова,
- тип F, E, A, V — на границу слова,
- тип H, S, Y — на границу полуслова,
- тип C, X, P, Z, B — выравнивание не происходит.

Больше никаких действий по оператору DC или DS с нулевой кратностью не выполняется: константа (или константы) не строятся, резервирование памяти не производится. Модификатор длины влияет на выравнивание (выравнивание в этом случае не производится) и на характеристику длины имени, которое может присутствовать в поле названия оператора. Присутствие или отсутствие других модификаторов и самих констант на результат не влияет. Разница между операторами DS и DC с нулевой кратностью заключается только в том, что байты, пропущенные при выравнивании по оператору DS, не изменяются, а байты, пропущенные при выравнивании по оператору DC, очищаются нулями.

Выполнение выравнивания на границу с помощью нулевой кратности очень широко используется при программировании на языке Ассемблера. Допустим, необходимо зарезервировать область памяти длиной 240 байт, которая должна начинаться на границе двойного слова, так как там будут находиться данные для команд, выполняющих действия над длинными числами с плавающей точкой. Зарезервировать такую область можно следующими операторами:

Название	Операция	Операнды
PLAD	DS DS	ØD CL24Ø

Первый оператор DS устанавливает счетчик адреса на границу двойного слова, а следующий резервирует 240 байт для области.

Оператор DS с нулевой кратностью может использоваться также для присваивания имени области памяти без фактического резервирования области. Операторы DS, следующие за таким оператором, могут использоваться для резервирования полей внутри этой области и присваивания им имен.

Предположим, записи длиной 80 байт вводятся в некоторую область памяти для обработки. Каждая запись имеет следующий формат:

- байты 1—4 не используются;
- байты 5—20 содержат фамилию рабочего;

байты 21—24 не используются;
 байты 25—30 содержат табельный номер;
 байты 31—36 содержат дату;
 байты 37—46 не используются;
 байты 47—54 содержат размер заработной платы;
 байты 55—80 не используются.

В программе выполняются действия как над всей записью в целом, так и над отдельными полями записи. Поэтому область можно определить с помощью оператора DS с нулевой кратностью, а потом разбить на поля с помощью других операторов DS.

Следующая последовательность команд показывает, как можно, используя операторы DS, присвоить имя области, отведенной для ввода записи, и определить поля внутри этой области:

Название	Операции	Операнды
BBOD	DS	ØCL8Ø
NAME	DS	CL4
	DS	CL16
TABN	DS	CL4
	DS	CL6
DATA	DS	ØCL6
DAY	DS	CL2
MEC	DS	CL2
GOD	DS	CL2
	DS	CL1Ø
SARPL	DS	CL8
	DS	CL26

Оператор DS с именем BBOD и нулевой кратностью не резервирует память, он только именуется всю область именем BBOD и определяет длину этой области, равную 80. При выполнении операции над всей областью можно использовать имя BBOD, значение которого будет равно адресу первого байта области, а характеристика длины — длине определяемой области.

Следующие за оператором BBOD операторы DS резервируют память для области, причем отдельно для каждого поля записи. Поля, которые не используются, резервируются неименованными операторами DS. Поля, которые используются в программе, резервируются с помощью именованных операторов DS. Оператор DS с именем DATA только именуется область памяти из шести байт, так как там указана нулевая кратность. Шесть байт этой области следующими операторами DS определяются как отдельные двухбайтовые поля этой области (DAY, MEC, GOD). Таким образом, при работе с записью в программе можно будет просто запрограммировать обращение к любому полю записи, так как каждое из них определено своим именем и характеристикой длины.

Команда Ассемблера EQU (ПРИСВОИТЬ ЗНАЧЕНИЕ) используется для определения символического имени путем присваивания ему значения, характеристики длины и переместимости, которые определяются выражением, записанным в поле операндов этого оператора. Оператор EQU имеет следующий формат:

Название	Операция	Операнды
Простое символическое имя, параметр или соединение параметра с другими знаками	EQU	Выражение

В поле названия записывается символическое имя, которому по оператору EQU присваиваются те же характеристики, которые имеет выражение в поле операндов. Выражение в поле операндов может быть абсолютным или простым переместимым. Все символические имена в выражении должны быть предварительно определены.

Значение символического имени из поля названия оператора EQU равно значению выражения, указанного в операнде этого оператора. Если значение выражения отрицательное, то за значение имени из поля названия принимаются 24 младших бита дополнительного кода результата. Характеристика длины имени оператора EQU будет равна характеристике длины выражения из поля операндов этого оператора. Она равна характеристике длины самого левого (или единственного) термина выражения. Если этот терм является значением счетчика адреса (*), самоопределенным термом или ссылкой на характеристику длины, то характеристика длины этих термов (а следовательно, и выражения) равна 1.

В приведенной ниже программе используются символические имена, указывающие номера регистров. Эти символические имена определены командой Ассемблера EQU:

Название	Операция	Операнды
BEGIN	BALR USING LA LR MVI	15,0 *,15 REG2,5 REG1,REG2 RES,OPER
RES	DS	X
REG1	EQU	2
REG2	EQU	3
OPER	EQU	X'3F'
	END	BEGIN

Символическим именам REG1, REG2 присваиваются абсолютные значения 2, 3, которые соответствуют номерам общих регистров. В команде MVI непосредственный операнд записан в виде символического имени OPER. Этому символическому имени оператором EQU присваивается значение X'3F'.

Рассмотрим еще один пример использования оператора EQU. Допустим, в программе используется одно и то же выражение. Можно оператором EQU приравнять это выражение к символическому имени и затем использовать это имя вместо выражения. Например, имеется следующая последовательность команд:

Название	Операция	Операнд
. . .	MVC	NAME+ALP-5(6),OBL
. . .	L	5,NAME+ALP-5
. . .	ST	5,NAME+ALP-5
OBL	DC	CL15'ABCDEF'
NAME	DS	CL100
ALP	EQU	20
	END	

Эту же последовательность команд можно записать следующим образом:

Название	Операция	Операнды
. . .	MVC	BIR(6),OBL
. . .	L	5,BIR
. . .	ST	5,BIR
OBL	DC	CL15'ABCDEF'
NAME	DS	CL100
ALP	EQU	20
BIR	EQU	NAME+ALP-5
	END	

3.10. ВОЗМОЖНОСТИ ВОЗДЕЙСТВИЯ НА ФОРМАТ ВЫВОДА РЕЗУЛЬТАТОВ ТРАНСЛЯЦИИ

Результатом работы транслятора является распечатка результатов трансляции и объектный модуль на картах (программа, которая после редактирования может выполняться на ЕС ЭВМ). Формат выводимых результатов трансляции в основном постоян-

ный. Но язык Ассемблера предоставляет программисту возможность оказывать некоторое влияние на формат результатов, выводимых транслятором Ассемблера. Для этой цели предназначены команды Ассемблера TITLE, EJECT, SPACE, PRINT, PUNCH и REPRO.

3.10.1. Управление выводом распечатки

Транслятор позволяет идентифицировать распечатку результатов трансляции и карты с объектным модулем. Для этого в программе на языке Ассемблера необходимо записать оператор TITLE.

Оператор TITLE (ИДЕНТИФИЦИРОВАТЬ ВЫВОД) указывает транслятору, какое заглавие печатать в начале каждой страницы распечатки результатов трансляции и как идентифицировать выводимые карты объектного модуля.

Оператор TITLE имеет следующий формат:

Название	Операция	Операнды
Любое символическое имя или пробел	TITLE	От 1 до 100 знаков, заключенных в апострофы

Правила записи поля названия в операторе TITLE несколько отличаются от правил записи этого поля в других операторах. Простое символическое имя, указанное в поле названия оператора TITLE, может содержать только от одной до четырех (а не до восьми) букв или цифр, причем они могут записываться в любой последовательности, первым знаком может быть цифра. Такие же требования накладываются на имена, которые будут получены в поле названия оператора TITLE после генерации, если там были записаны параметры или соединение параметров со знаками. В поле операндов оператора TITLE можно записывать до 100 любых знаков кода ДКОИ. Последовательность этих знаков должна быть заключена в апострофы. Знак & и апостроф внутри этих апострофов должны быть представлены двумя знаками & и двумя апострофами.

Исходный модуль может содержать несколько операторов TITLE. В этом случае только первый оператор TITLE может иметь в поле названия любое символическое имя, в остальных операторах TITLE допускается только символическое имя перехода или пробел.

Оператор TITLE вызывает вывод новой страницы распечатки, т. е. после этого оператора текст исходного и объектного модулей будет печататься, начиная с новой страницы. Кроме того, текст из поля названия первого оператора TITLE будет выво-

даться в заглавии на каждой странице распечатки и перфорироваться в колонках 73—76 каждой карты объектного модуля. Исключение составляют те карты, которые выводятся с помощью команд Ассемблера REPRO и PUNCH. Текст из поля операндов, записанный внутри апострофов, будет выводиться как заглавие на каждой странице распечатки за данным оператором TITLE до тех пор, пока не встретится новый оператор TITLE.

Предположим, что программа, написанная на языке Ассемблера, вычисляет выражение. В начале программы можно записать следующий оператор:

Название	Операция	Операнды
BVVV	TITLE	'ВЫЧИСЛЕНИЕ ВЫРАЖЕНИЙ'

После трансляции этой программы на картах с объектным модулем (колонки 73—76) будет отперфорировано BVVV, в начале каждой страницы распечатки текста объектного и исходного модулей появится следующее заглавие:

BVVV ВЫЧИСЛЕНИЕ ВЫРАЖЕНИЙ

Страницы распечатки результатов трансляции, содержащие другую информацию (словарь внешних имен, словарь переместимых адресных констант, таблицу перекрестных ссылок, список диагностических сообщений об ошибках, текст до первого оператора TITLE), в заглавии будут иметь только содержимое поля названия первого оператора TITLE.

Допустим, программа вычисления выражения написана так, что она имеет ряд отдельных подпрограмм, выполняющих определенные функции, например поиск разделителя, обработку десятичного числа. В этом случае в программе можно записать несколько операторов TITLE, озаглавив отдельно каждую подпрограмму, как это показано ниже.

Название	Операция	Операнды
BVVV	TITLE	'ВЫЧИСЛЕНИЕ ВЫРАЖЕНИЙ'
• • •	TITLE	ПОДПРОГРАММА: ПОИСК РАЗДЕЛИТЕЛЯ'
• • •	TITLE	ПОДПРОГРАММА: ОБРАБОТКА ТЕРМА'
OSHI	TITLE	ПОДПРОГРАММА: ABC&&D'E'

Каждый приведенный оператор TITLE находится в начале подпрограммы, поэтому текст каждой подпрограммы будет выводиться на печать, начиная с новой страницы. На каждой странице распечатки текста, следующего за первым оператором TITLE до второго оператора TITLE, будет печататься заглавие:

BVVV ВЫЧИСЛЕНИЕ ВЫРАЖЕНИЙ

После второго оператора TITLE каждая страница распечатки будет иметь следующее заглавие:

BVVV ПОДПРОГРАММА: ПОИСК РАЗДЕЛИТЕЛЯ

Все карты объектного модуля будут иметь идентификатор BVVV в колонках 73—76 независимо от присутствия нескольких операторов TITLE.

В приведенном примере последний оператор TITLE тоже имеет имя в поле названия. Но это имя не будет появляться ни в распечатке, ни на картах, а оператор TITLE будет отмечен как ошибочный. Однако текст из поля операндов этого оператора будет печататься на следующих страницах распечатки как заглавие, причем в сочетании с именем первого оператора TITLE. Таким образом, в распечатке после последнего оператора TITLE будет печататься следующее заглавие:

BVVV ПОДПРОГРАММА: ABC&D'E

Знаки & и апостроф, которые должны появиться в заглавии, в операнде оператора TITLE записаны как два знака & и два апострофа.

Кроме идентификации распечатки результатов трансляции, можно при необходимости вызвать печать распечатки текста исходного и объектного модулей в определенном месте с новой страницы или вставить в распечатку текста пустые строки. Эти действия выполняются транслятором по операторам EJECT и SPACE.

Рассмотрим следующую задачу: необходимо выполнить ввод информации, ее обработку и вывод информации. Обработка информации включает два этапа: перекодировку и редактирование. В программу, реализующую поставленную задачу, можно включить операторы, которые будут управлять выводом распечатки при работе транслятора. Можно, например, три части текста программы, выполняющие ввод, обработку и вывод соответственно, печатать, начиная каждую с новой страницы, а часть программы, выполняющую два этапа обработки, разделить, например, четырьмя пустыми строками.

Программа в этом случае выглядела бы следующим образом (начало каждой части отмечено оператором комментариев):

Название	Операция	Операнды
** BEG	BALR USING	ввод 15,0 *,15
. . . ** **	EJECT	. . . обработка перекодировка
. . . **	LA	1,2
. . . **	SPACE	. . . 4 редактирование
. . . **	EJECT	. . .
. . .	END	вывод BEG

Распечатка текста первой части программы (ввода) будет выводиться с начала страницы, так как это начало текста. Перед командами, выполняющими обработку, в программе записан оператор EJECT, который вызывает печать следующего за ним текста с новой страницы.

Оператор EJECT имеет следующий формат:

Название	Операция	Операнды
Символическое имя перехода или пробел	EJECT	Не используется

Если строка распечатки, расположенная после оператора EJECT, будет печататься первой на странице независимо от того, предшествует ей EJECT или нет, то оператор EJECT не окажет никакого действия на вывод распечатки. Если после EJECT появляется еще один или несколько операторов EJECT, то пропускается одна или несколько страниц распечатки, т. е. напечатаются страницы только со строкой заглавия. Таким образом, первый оператор EJECT, записанный в приведенной программе, вызовет печать следующего за ним текста программы с новой страницы.

Предполагается, что в приведенной программе оператор SPACE располагается после всех операторов программы, выполняющих первый этап обработки — перекодировку.

Оператор SPACE имеет следующий формат:

Название	Операция	Операнды
Символическое имя перехода или пробел	SPACE	Десятичное число или пробел

Десятичное число, записанное в поле операндов оператора SPACE, которое не должно быть больше 255, указывает количество пустых строк, которое должно появиться в распечатке. Если в качестве операнда SPACE используется пробел, то в распечатке появится одна пустая строка. Если десятичное число из поля операндов SPACE превышает число свободных строк на данном листе распечатки, то оператор SPACE вызовет те же действия, что и оператор EJECT. Таким образом, записанный в приведенном примере оператор SPACE, вызовет появление в распечатке четырех пустых строк, а так как этот оператор расположен после операторов, выполняющих перекодировку, но перед операторами, выполняющими редактирование, то в распечатке текст этих частей программы будет разделен четырьмя пустыми строками.

Второй оператор EJECT находится перед частью программы, выполняющей вывод. Поэтому третья часть программы будет печататься, начиная с новой страницы. Операторы TITLE, SPACE, EJECT сами не появляются в распечатке результатов трансляции.

3.10.2. Воздействие на содержание распечатки

Программист может управлять содержанием той части распечатки результатов трансляции, которая содержит текст исходного и объектного модулей. Для этой цели используется оператор PRINT (УПРАВЛЯТЬ ПЕЧАТЬЮ). На другие части распечатки (словарь внешних имен, словарь переместимых адресных констант, таблица перекрестных ссылок, сообщения об ошибках) оператор PRINT не влияет. С помощью оператора PRINT можно отменить печать некоторой информации, не представляющей интереса в данной распечатке, и таким образом сократить время, затрачиваемое на выполнение трансляции. Оператором PRINT можно отменить вывод на печать следующей информации:

- всего или части текста исходного и объектного модулей;
- операторов, которые сгенерированы по макрокомандам;
- части объектного кода констант.

Последующим использованием оператора PRINT можно восстановить режим вывода информации на печать.

Оператор PRINT имеет следующий формат:

Название	Операция	Операнды
Символическое имя пере- хода или пробел	PRINT	От одного до трех операндов, раз- деленных запятыми, указывающих режимы печати

В поле операндов можно указать следующие операнды:

- а) ON — печатать последующий текст исходного и объектного модулей

или

OFF — не печатать последующий текст исходного и объектного модулей;

б) GEN — печатать операторы, сгенерированные по макрокомандам, в последующем тексте программы

или

NOGEN — не печатать операторы, сгенерированные по макрокомандам, в последующем тексте программы. Сами макрокоманды печатаются, а также печатаются сообщения, выдаваемые по операторам MNOTE, присутствующим в макроопределении;

в) DATE — печатать весь объектный код констант в последующем тексте программы.

или

NODATE — печатать только первые 8 байт константы в последующем тексте программы.

Рассмотрим пример программы, использующей оператор PRINT.

Название	Операция	Операнды
	BALR	15,0
	USING	*,16
. . .	READ	. . .
NAME	DC	B
. . .		10X'1234567890'
NAME1	PRINT	DATE
	DC	10X'12345'
	PRINT	NODATE,NOGEN
	READ	A
. . .	PRINT	OFF,DATE,GEN
. . .	PRINT	DATE,ON
. . .	END	. . .

Как видно из приведенной программы, одна программа может содержать любое число операторов PRINT. Режим, установленный каждым оператором PRINT, остается в силе, пока не встретится другой оператор PRINT, отменяющий этот режим. В начале приведенной программы нет операторов PRINT. В этом случае действует стандартный режим для вывода распечатки текста исходного и объектного модулей, соответствующий следующему оператору PRINT:

Название	Операция	Операнды
	PRINT	ON,NODATE,GEN

Таким образом, начальная часть текста исходного и объектного модулей будет выводиться на печать (режим ON), для констант будет печататься только 8 байт (режим NODATE), операторы, сгенерированные по макрокомандам, будут печататься (режим GEN). Предположим, READ — это макрокоманда. Тогда будут отпечатаны операторы, которые генерируются по этой макрокоманде, и отмечены на печати знаком +. Объектный код константы с именем NAME будет отпечатан не полностью (первые 8 байт).

Первый оператор PRINT, присутствующий в программе, содержит только один операнд — DATE. Если в операторе PRINT какой-либо из операндов не указывается, то действует режим, определенный последним из предыдущих операторов PRINT, в котором этот операнд присутствует. Таким образом, по первому оператору PRINT в примере режимы ON и GEN остаются в силе, а режим NODATE изменяется на DATE. Для константы с именем NAME1 будут распечатаны все 50 байт.

Второй оператор PRINT, записанный в программе, отменяет режимы DATE и GEN. Режим ON остается в силе. Режимы, установленные вторым оператором PRINT, обозначают, что операторы, сгенерированные по второй макрокоманде READ, печататься не будут (режим NOGEN) и для констант опять будут печататься только 8 байт их объектного кода.

В третьем операторе PRINT указаны противоречивые режимы вывода: не выводить распечатку текста, но печатать операторы, сгенерированные по макрокомандам, и весь объектный код констант. В таких случаях режимы устанавливаются по следующим правилам:

режим OFF подавляет режимы GEN и DATE;

режим NOGEN подавляет режим DATE.

Таким образом, третий оператор PRINT установит режим: не печатать последующий текст программы. Режимы GEN, DATE, хотя и будут установлены, но во время печати будут подавлены режимом OFF.

Последний оператор PRINT в программе указывает режимы печати текста программы и всего объектного кода констант. В этом случае вступит в силу и режим GEN, установленный предыдущим оператором PRINT. Из этого же оператора видно, что операнды в операторе PRINT можно записывать в любом порядке.

3.10.3. Вывод информации, дополнительной к объектному модулю

В программе на языке Ассемблера можно предусмотреть включение в объектный модуль некоторой дополнительной информации. Для этого предназначены команды Ассемблера PUNCH и REPRO. Транслятор во время вывода объектного модуля включает в него информацию, указываемую этими операторами.

Оператор PUNCH имеет следующий формат:

Название	Операция	Операнды
Символическое имя перехода или пробел	PUNCH	От 1 до 80 знаков, заключенных в апострофы

По оператору PUNCH (ПЕРФОРИРОВАТЬ КАРТУ) производится перфорация карты, содержимое которой определяется полем операндов этого оператора. По оператору PUNCH перфорируется одна карта. В программе может быть несколько операторов PUNCH. В операнде оператора PUNCH записывается последовательность любых знаков кода ДКОИ, заключенная в апострофы. Для представления в операнде знака & или апострофа необходимо записывать два апострофа или два знака &. На карту выводится один апостроф или один знак &. Если количество знаков в операнде больше 80, то на карту перфорируются первые 80 знаков, остальные знаки игнорируются, а оператор отмечается как ошибочный. Перфорация знаков из поля операндов начинается с первой колонки карты. Параметры, встречающиеся в поле операндов оператора PUNCH, заменяются их значениями. Если оператор PUNCH встречается перед операторами, влияющими на значение счетчика адреса, то перфорируемая по нему карта выводится перед всеми картами объектного модуля. В других случаях перфорируемая карта будет выводиться в том месте, где записан оператор PUNCH.

Вывод информации, дополнительной к объектному модулю, можно также выполнить, записав в программе оператор REPRO.

Оператор REPRO имеет следующий формат:

Название	Операция	Операнды
Символическое имя перехода или пробел	REPRO	Не используется

По оператору REPRO (ПЕРФОРИРОВАТЬ СЛЕДУЮЩУЮ КАРТУ) перфорируется карта, содержимое которой определяется строкой программы, следующей за оператором REPRO. В этой строке, начиная с первой позиции строки и кончая 80-й позицией, могут быть записаны любые знаки кода ДКОИ. Первая позиция строки будет соответствовать первой колонке выводимой карты. Перфорируемая последовательность знаков не проверяется на присутствие в ней параметров, и если они там записаны, то подстановка их значений в строку не производится. Если нужно отперфорировать знак апостроф или знак &, то в строке, следующей за оператором REPRO, нужно записать соответственно только один апостроф или знак &.

Рассмотрим случай использования оператора REPRO. Допустим, необходимо после трансляции поместить объектный модуль в библиотеку объектных модулей. Можно задать перфорацию управляющего оператора Библиотекаря CATALR с именем модуля, с которого должен начинаться каталогизируемый модуль, с помощью оператора REPRO. Пусть объектный модуль необходимо каталогизировать в библиотеку под именем NAME. Тогда программа будет выглядеть следующим образом:

Название	Операция	Операнды
NASCALO	REPRO	NAME
...	CATALR	5,0
...	BALR	MOD
...	COPY	...
	END	

Оператор REPRO записан перед операторами, влияющими на значение счетчика адреса. Поэтому содержимое следующей за ним строки будет отперфорировано перед всеми картами объектного модуля. Вывод карты по оператору REPRO выполнится на то же устройство, куда будет выводиться объектный модуль.

В этом примере используется оператор COPY. Этот оператор дает возможность работать с исходными модулями, написанными на языке Ассемблера и хранящимися в библиотеке исходных модулей. Эти модули, которые обычно являются готовыми программами на языке Ассемблера, можно включать в исходную программу.

Оператор COPY имеет следующий формат:

Название	Операция	Операнды
Пробел	COPY	Одно простое символическое имя

По оператору COPY (КОПИРОВАТЬ КНИГУ) из подбиблиотеки исходных модулей Ассемблера включается в программу книга, состоящая из последовательности операторов на языке Ассемблера. Имя, записанное в поле операндов, является именем включаемой книги. Транслятор по оператору COPY включает исходные операторы книги непосредственно за этим оператором COPY. Операторы книги затем будут протранслированы вместе с программой, как если бы эти исходные операторы были написаны в этом месте программы. Вызываемая книга не должна содержать операторов COPY, END, PRINT, ICTL, ISEQ, MACRO и MEND.

Если встречаются операторы COPY с одинаковым именем в поле операндов, то книга включается в модуль каждый раз. Копируемый текст всегда имеет стандартный формат, который не может быть изменен оператором ICTL.

В нашем примере в программу будут вставлены исходные операторы на языке Ассемблера, которые в библиотеке исходных модулей составляют книгу с именем MOD.

3.11. СЕКЦИОНИРОВАНИЕ И СОЕДИНЕНИЕ ПРОГРАММ

Программы на языке Ассемблера, реализующие решение больших задач, обычно получаются длинными и сложными. В разработке таких программ могут участвовать несколько человек. Поэтому возникает необходимость писать большие программы по частям, а потом соединять эти части в одну выполняемую программу.

Процесс разделения программы на части называется секционированием. Для осуществления секционирования и дальнейшего соединения отдельных частей программы язык Ассемблера предоставляет следующие возможности:

- с помощью операторов START и CSECT можно разбить программу на отдельные части, называемые программными секциями;

- с помощью оператора COM можно определить общую область программы, к которой могут обращаться все части этой программы;

- с помощью оператора DSECT можно описывать области памяти из других программ без их фактического резервирования в данной программе;

- можно указывать отдельные символические имена в данной исходной программе, которые могут использоваться в других программах. Для этого предназначены операторы ENTRY, CSECT, START;

- можно указывать отдельные символические имена из других программ, используемые в данной программе. Это выполняется с помощью оператора EXTRN и констант типа V.

Процесс подготовки задачи к выполнению можно разделить на три основных этапа: составление программы (кодирование), трансляция и редактирование. Рассмотрим использование программных секций отдельно на каждом этапе.

На этапе кодирования вся программа разбивается на части — программные секции. В программную секцию нужно выделять часть программы, выполняющую некоторое законченное действие. Например, можно выделять в секцию подпрограмму или блок команд, который можно целиком использовать в другой программе. Разбиение на секции в основном определяется требованиями третьего этапа — редактирования. Операторы каждой программной секции в памяти будут расположены в таком порядке, в каком они записаны в программе. Но положение программных секций отно-

сительно друг друга может изменяться и окончательно определяется Редактором. Поэтому при программировании необходимо предусмотреть правильные связи между программными секциями.

Установление связей между секциями зависит от того, транслируются они вместе или отдельно друг от друга. В любом случае для каждой программной секции при кодировании должен определяться свой регистр базы. Если в секции используются символические имена, определенные в другой секции, которая транслируется отдельно, то эти имена нужно определить с помощью оператора EXTRN. Значения таких имен можно использовать также с помощью констант типа V. В свою очередь имена, используемые в других отдельно транслируемых секциях, должны указываться операторами ENTRY, CSECT или START в тех секциях, в которых они определяются. Если несколько секций будут транслироваться вместе, то в каждой секции можно использовать имена из любой другой секции, транслируемой вместе, при этом определять эти имена операторами EXTRN или ENTRY не требуется.

Таким образом, если секции программы транслируются отдельно, операторами START, CSECT, EXTRN или ENTRY указывается информация, необходимая для связи отдельно транслируемых секций программы. Кроме того, информацию, используемую несколькими секциями, можно подготовить в общей области, которая определяется оператором COM. Имеется возможность с помощью оператора DSECT описать структуру области памяти, которая зарезервирована в одной секции и используется в другой секции.

На этапе трансляции обрабатываемой единицей является исходный модуль. В исходный модуль может быть включена одна или более программных секций. Каждый исходный модуль, подготовленный для трансляции, должен заканчиваться оператором END, который указывает транслятору на конец исходного модуля. Формат оператора END следующий:

Название	Операция	Операнды
Символическое имя перехода или пробел	END	Простое переместимое выражение или пробел

Операнд оператора END указывает точку модуля, с которой начинается выполнение данной части программы.

В исходном модуле, подготовленном для трансляции, все символические имена должны быть уникальными.

Каждый исходный модуль при трансляции преобразуется в объектный модуль. В объектный модуль наряду с текстом программных секций и словарем переместимых адресных констант входит также словарь внешних имен. В словаре внешних имен представлена вся информация, необходимая для установления связей между секциями, поставляемая операторами START, CSECT, COM, ENTRY, EXTRN и константами типа V. Транслятор не про-

веряет, является ли информация, предоставленная для связи между секциями, правильной и достаточно полной. Такого рода ошибки выявляются на этапе редактирования.

На этапе редактирования распределяется память для программных секций, транслируемых вместе или отдельно, и формируется один или несколько сегментов (фаз) выполняемой программы. Входом для Редактора являются объектные модули или отдельные программные секции из объектных модулей. Для установления связей между секциями Редактор использует словарь внешних имен и информацию о распределении памяти для программных секций. После редактирования полученную программу можно выполнять на ЭВМ.

Деление программы на секции не обязательно, многие программы могут быть написаны без использования команд секционирования. В программе не нужно также использовать средства символического соединения, если эта программа не имеет никаких связей с другими программами.

3.11.1. Деление на программные секции

Программная секция — это блок команд, имя которого определяется с помощью оператора CSECT или START. Каждая программная секция имеет свой признак переместимости. Символические имена и счетчик адреса характеризуются признаком переместимости той программной секции, к которой они принадлежат.

В общем программную секцию можно рассматривать как блок команд, который можно без потери его работоспособности включать в любой сегмент одной программы и в другую программу.

Обычно программная секция идентифицируется оператором CSECT. Но если желательно указывать начальный адрес программы, то идентификацию первой программной секции можно выполнить оператором START.

Как отмечалось в 3.5, оператор START указывает начальное значение счетчика адреса для исходного модуля. Если исходный модуль состоит из нескольких программных секций, то оператор START указывает начальное значение счетчика адреса для первой программной секции модуля. Кроме того, символическое имя из поля названия оператора START является именем первой программной секции. Это имя является переместимым символическим именем, значение его равно адресу первого байта программной секции. Характеристика длины имени равна 1. Все операторы, следующие за оператором START, транслируются как часть этой первой программной секции. Это продолжается до тех пор, пока в транслируемом модуле не встретятся операторы CSECT, DSECT или COM. Эти операторы указывают транслятору, что данная транслируемая секция пока закончена (секция в модуле может иметь продолжение), далее следует другая программная секция, фиктивная или общая область.

Команда Ассемблера CSECT (ОПРЕДЕЛИТЬ ПРОГРАММНУЮ СЕКЦИЮ) определяет начало или продолжение любой

программной секции. Оператором CSECT можно определять и начало первой секции в модуле, но в этом случае начальное значение счетчика адреса будет всегда равно нулю.

Оператор CSECT имеет следующий формат:

Название	Операция	Операнды
Любое символическое имя или пробел	CSECT	Не используется

Если в поле названия оператора CSECT записано символическое имя, то оно является именем программной секции. Все операторы, следующие за оператором CSECT, будут принадлежать к этой программной секции, пока не встретится оператор CSECT, определяющий другую секцию, или оператор DSECT, определяющий фиктивную область, или оператор COM, определяющий общую область.

В одном и том же модуле может появиться несколько операторов CSECT с одним и тем же именем, но это означает не повторное определение имени, а указание на продолжение секции с таким именем. Первый оператор CSECT с некоторым именем определяет начало секции, а остальные операторы CSECT с таким же именем — продолжение этой секции. Если программную секцию определяет оператор START с некоторым именем, то операторы CSECT с таким же именем определяют продолжение этой секции. Таким образом, операторы различных секций в исходном модуле могут быть перемешаны, но транслироваться они будут правильно: операторам, принадлежащим к одной программной секции, присваиваются последовательные адреса памяти.

Транслятор формирует счетчик адреса для каждой программной секции. Начальное значение счетчика адреса первой программной секции устанавливается оператором START или принимается равным нулю, если оператор START отсутствует. Счетчик адреса каждой следующей секции начинается непосредственно за последним использованным адресом предыдущей секции. Порядок секций в объектном модуле зависит от того, в каком порядке появились первые операторы секций. Каждая программная секция, следующая за предыдущей, начинается со следующего двойного слова.

В приведенном ниже примере определяются три программные секции. Оператор START определяет первую программную секцию и называет ее именем CEK. Все операторы, следующие за оператором START до оператора CSECT с именем CEK1, будут отнесены к первой секции. Начальный адрес секции CEK, указанный в операнде оператора START, равен X'1000'.

Название	Операция	Операнды
CEK	START	X'1000'
CEK1	CSECT	...
NAME	LR	3,5
CEK2	CSECT	...
DS1	DSECT	...
CEK1	CSECT	продолжение секции CEK1
...	ORG	NAME—50
...	ORG	...
CEK	CSECT	продолжение секции CEK
CEK2	CSECT	продолжение секции CEK2
...	END	BEGIN

Оператор CSECT с именем CEK1 начинает новую секцию. Начальным значением счетчика адреса этой секции является адрес первого двойного слова, которое следует за секцией CEK, учитывая все ее продолжения. Все операторы, следующие за оператором CSECT до оператора CSECT с именем CEK2, будут принадлежать секции CEK1.

Секция CEK2 будет прервана оператором DSECT. Оператор DSECT определяет фиктивную область. Подробно она рассматривается в 3.11.3. Операторы, следующие за оператором DSECT, уже не относятся к секции с именем CEK2.

Далее следует оператор CSECT с именем CEK1. Этим оператором определяется продолжение секции CEK1. Счетчик адреса для продолжения будет изменяться, начиная с того значения, которое было последним для первой части секции CEK1.

Оператор CSECT с именем CEK прерывает продолжение секции CEK1 и определяет продолжение первой секции модуля.

Далее в модуле следует продолжение секции CEK2. Оператор END заканчивает модуль. В поле операндов оператора END записано символическое имя BEGIN, определяющее точку программы, с которой начинается ее выполнение.

В данном примере начальное значение счетчика адреса секции CEK равно X'1000'. Начальное значение счетчика адреса секции CEK1 равно адресу двойного слова, следующего непосредственно за последним байтом, использованным секцией CEK, учитывая все продолжения секции CEK. Таким образом, начальное значение счетчика адреса секции CEK1 равно X'1000'+L1+L2, где X'1000'—начальный адрес первой секции CEK, установленный оператором START, L1—длина первой части секции CEK, L2—

длина второй части секции CEK. Начальный адрес секции CEK2 будет определен как начальный адрес секции CEK1 плюс длина двух частей секции CEK1, записанных в исходном модуле. Кроме того, начальный адрес каждой секции выравнивается на границу двойного слова. Таким образом, несмотря на то, что в исходном модуле секции разорваны и части их перемешаны, в основной памяти каждая секция модуля будет располагаться в непрерывной области памяти и все секции будут следовать одна за другой: CEK, CEK1, CEK2.

Отметим некоторые свойства оператора ORG, которые не были рассмотрены в 3.5. В поле операндов оператора ORG записывается простое переместимое выражение. В этом выражении неспаренное переместимое имя должно быть определено в той же программной секции, в которой появился оператор ORG. Например, в приведенном модуле оператор ORG используется в секции CEK1. Переместимое символическое имя NAME определяется в этой же секции. Оператор ORG уменьшает значение счетчика адреса. Необходимо следить за тем, чтобы значение NAME-50 не оказалось меньше начального адреса программной секции CEK1, так как в этом случае оператор ORG будет ошибочен. В секции CEK1 далее используется еще один оператор ORG, операнд которого опущен. В этом случае счетчику адреса присваивается значение, которое равно адресу байта, следующего за последним байтом, использованным данной секцией к моменту обработки оператора ORG без операнда. Таким образом, если предыдущий оператор ORG уменьшил значение счетчика адреса для присвоения других имен областям памяти в программной секции CEK1, то для восстановления значения счетчика адреса к его максимальному значению в этой секции используется оператор ORG без операнда.

В исходном модуле особое место занимает первая программная секция в модуле. Она имеет следующие свойства:

начальное значение ее счетчика адреса можно определить оператором START;

она содержит литералы, которые записаны между последним оператором LTORG, имеющимся в программе, и оператором END, или все литералы, если в исходном модуле вообще нет операторов LTORG.

В приведенном модуле для первой программной секции оператором START определяется начальный адрес X'1000'. Если в этом модуле используются литералы и нет операторов LTORG, то они будут размещены транслятором после всех операторов секции CEK, а только после литералов будут располагаться операторы секции CEK1. В приведенном модуле вместо оператора START можно было использовать оператор CSECT с именем CEK. В этом случае начальное значение счетчика адреса первой секции (начальный адрес модуля) было бы установлено равным нулю.

В поле названия оператора START или CSECT может отсутствовать символическое имя. Если программная секция модуля

определяется неименованным оператором START или CSECT, то такая секция называется неименованной. В модуле может быть только одна неименованная программная секция. Любой последующий неименованный оператор CSECT определяет продолжение неименованной программной секции. Началом неименованной программной секции считается также начальная часть модуля, которая не определяется оператором START или CSECT. Модуль, который вообще не содержит операторов START и CSECT, считается неименованной программной секцией. Рассмотрим следующий исходный модуль:

Название	Операция	Операнды
	BALR	15,0
	USING	*,15
	LA	4,100
...
CEK1	CSECT	3,4
...	CSECT	...
*		продолжение неименованной секции
...	END	...

Все исходные операторы до оператора CSECT с именем CEK1 будут принадлежать к неименованной программной секции. Она будет прервана операторами секции CEK1. Появляющийся затем оператор CSECT без имени в поле названия будет определять продолжение неименованной программной секции. В операторе CSECT комментарии можно записывать сразу после поля операции, оставляя хотя бы один пробел, так как поле операндов в этом операторе не используется.

3.11.2. Определение регистров базы в многосекционной программе

Для того чтобы транслятор Ассемблера правильно представил неявные адреса в виде регистра базы и смещения при наличии в исходном модуле нескольких секций, необходимо соблюдать следующие правила:

для каждой программной секции оператором USING должен быть определен регистр базы и базовый адрес;

базовый адрес, определенный оператором USING для базирования неявных адресов данной программной секции, должен принадлежать к этой же секции;

оператор USING, определяющий базовый адрес и регистр базы для базирования неявных адресов, должен предшествовать всем операторам, использующим эти адреса в поле операндов. Этот оператор может находиться не в той программной секции, к которой принадлежит базовый адрес;

если исходный модуль содержит несколько программных секций, имеющих продолжение, то желательно назначать разные регистры базы для каждой секции. Если же регистры базы для этих секций должны быть одинаковыми, то в каждом продолжении секции необходимо заново определить регистр базы с помощью оператора USING;

для абсолютных адресов регистры базы определяются независимо от секции;

регистры базы каждой секции должны своевременно загружаться необходимыми значениями с помощью машинных команд;

если в модуле во всех секциях используются литералы, принадлежащие к первой секции модуля (после последнего оператора LTORG в программе используются литералы), то необходимо позаботиться о сохранении содержимого регистра базы первой секции модуля при работе других секций, так как адреса литералов, используемых в других секциях, базируются этим регистром.

Рассмотрим определение и загрузку регистров базы в много-секционной программе на следующем примере:

Название	Операция	Операнды	
CEK1	START	X'1200'	начало секции CEK1
B	BALR	4,0	загрузка регистра базы
*	USING	*,4	назначение регистра базы для секции CEK1
	L	10,NAME	NAME базируется регистром 4
	LA	11,100	
	AR	10,11	
	L	5,BAS2	загрузка регистра базы для секции CEK2, BAS2 базируется регистром 4
*			
*	A	10,NAME1	NAME1 базируется регистром 4
	ST	10,NAME2	NAME2 базируется регистром 4
	USING	CEK2,5	назначение регистра базы для секции CEK2
*			
C1	LA	6,BAS2	загрузка регистра базы 6 для секции CEK1, BAS2 базируется регистром 4
*			
R4	MVC	C1A(5),C2A	C1A базируется регистром 4, C2A — регистром 5
*			
PB1	LA	11,C1	C1 базируется регистром 4
	B	C2	C2 базируется регистром 5
BAS2	DC	A(CEK2)	константа для регистра базы секции CEK2
*			
CEK2	CSECT		начало секции CEK2
C2	MVC	C1A+10(5),C2B	C2B базируется регистром 5, C1A+10 — регистром 4
*			
HET3	LA	10,C3	для C3 не определен регистр базы
	USING	BAS2,6	назначается регистр базы для части секции CEK1
*			
LIT	L	10,=F'200'	

Название	Операция	Операнды	
PB2	LA	11,C1	C1 базируется регистром 4
CEK3	CSECT		начало секции CEK3
	BALR	4,Ø	загрузка и назначение
	USING	*4	регистра базы для секции CEK3
R4B3	MVC	C3,C2A	C3 базируется регистром 4,
*			C2A — регистром 5
	L	1Ø,C1B	C1B базируется регистром 6
	LA	11,C1	для C1 не определён регистр базы
	B	C11	C11 базируется регистром 6
C3	DS	CL5	
CEK1	CSECT		продолжение CEK1
	USING	B+2,4	вторичное определение и
C11	L	4,BAS1	загрузка регистра базы 4
*			для секции CEK1,
*			BAS1 базируется регистром 6
R6	MVC	C1A,C2B	C1A базируется регистром 6,
*			C2B базируется регистром 5
			C1 базируется регистром 4
			для C3 не назначен регистр базы
	LA	12,C1	
	MVC	C3,BAS2	
	SVC	14	
NAME	DC	F'1Ø'	
NAME1	DC	F'2Ø'	
NAME2	DS	F	
BAS1	DC	A(B+2)	
C1A	DS	CL2Ø	
C1B	EQU	BAS1	
CEK2	CSECT		продолжение секции CEK2
C2A	DC	C'PAGE'	
C2B	DC	CL5'PRIN'	
	END	B	

Приведенный модуль имеет три программных секции, имена которых CEK1, CEK2, CEK3. Секции CEK1 и CEK2 имеют продолжение.

Для первой секции CEK1 оператор USING определяет регистр базы 4 и базовый адрес, равный B+2 из этой же секции. Здесь же командой BALR выполняется загрузка регистра базы тем базовым адресом, который указан транслятору оператором USING. Неявные адреса из секции CEK1, использованные в поле операндов любой секции модуля, будут базироваться регистром 4 до тех пор, пока не появится новый оператор USING с базовым адресом этой секции. Оператор USING, определяющий регистр базы 6, указывает базовый адрес BAS2, принадлежащий секции CEK1. После этого оператора для символических имен из секции CEK1 будут определены два регистра базы: 4 и 6. Тогда для каждого имени из секции CEK1, используемого в поле операндов, будет выбираться доступный регистр базы.

Таким образом, символические имена из секции CEK1, которые определены до адреса BAS2 (например, B, C1), базируются

регистром 4, так как только в этом регистре определен доступный для них базовый адрес (значение базового адреса в регистре 6 больше значения этих адресов). Имена секции CEK1, определенные после адреса BAS2 (например, C1A, C1B), базируются регистром 6, так как базовый адрес, указанный для этого регистра базы, дает наименьшее смещение. Например, в операторе с именем R4 символическое имя C1A из секции CEK1 базируется регистром 4 (регистр 6 еще не определен), а в операторе с именем R6 это символическое имя базируется регистром 6. Символическое имя C1, используемое в операторах с именами RB1 и RB2, оба раза базируется регистром 4, хотя перед оператором RB2 уже определены два регистра базы (4 и 6) для секции CEK1. Но регистр 6 недоступен для адреса C1, потому что базовый адрес, находящийся в регистре 6, больше адреса C1.

Оператор определения регистра базы может находиться не только в той секции, для которой она определяет регистр базы, но и в любой другой секции, в которой используются эти адреса. Это можно увидеть в приведенной программе на примере определения регистра базы для секции CEK2. Для секции CEK2 регистром базы назначен регистр 5. Определение регистра базы для секции CEK2 оператором USING выполняется в секции CEK1, предшествующей секции CEK2, так как в секции CEK1 уже используются символические имена C2A и C2, определяемые в секции CEK2. Необходимо, чтобы для них был определен регистр базы и базовый адрес до того, как они используются в командах. Загрузка этого регистра тоже выполняется в секции CEK1 до того, как используется имя из секции CEK2, причем именно тем адресом, который указан как базовый в операторе USING. Таким образом, не имеет значения, в какой секции появится оператор USING, определяющий регистр базы для данной секции. Главное, чтобы он появился в модуле до того, как используются неявные адреса из этой секции.

Для секции CEK3 регистром базы определяется общий регистр 4, который является также регистром базы для секции CEK1. Символические имена из секции CEK3 (например, C3) будут базироваться регистром 4. Но это лишь те имена, которые используются в поле операндов операторов, расположенных после оператора USING для секции CEK3. Например, для имени C3 к моменту обработки оператора с именем HET3 еще не найдется доступного регистра базы (с базовым адресом из секции CEK3), а при обработке оператора с именем R4B3 для имени C3 уже будет доступным регистр 4. Но необходимо заметить, что в этом случае регистр 4 станет уже недоступен для имен из секции CEK1. Если имена C1B и C1I из секции CEK1, используемые в секции CEK3, базируются регистром 6, то для имени C1 доступного регистра базы не будет (базовый адрес в регистре 6 больше адреса C1, а другого регистра базы, кроме регистра 6, с базовым адресом из секции CEK1 нет). Таким образом, при использовании одних и тех же регистров в качестве регистров базы в нескольких

секциях необходимо следить, правильно ли базируются в этих секциях неявные адреса из других секций.

После секции СЕК3 следует продолжение секции СЕК1. В начале продолжения регистр 4 снова определяется как регистр базы секции СЕК1. В продолжении секции СЕК1 один из операторов использует имя, определяемое в секции СЕК3. Для этого имени не будет доступного регистра базы, так как к моменту обработки этого оператора ни один из регистров, указанных как регистр базы, не будет содержать базовый адрес из секции СЕК3. Если бы в секции СЕК3 не использовались неявные адреса из секции СЕК1, а имена из секции СЕК3 не использовались в других секциях, то использование регистра 4 как регистра базы в двух секциях (СЕК1 и СЕК2) было бы правильным.

В приведенном примере в команде с именем LIT используется литерал. В модуле нет операторов LTORG. Значит, литерал после трансляции будет расположен в конце первой программной секции СЕК1. До команды с именем LIT, расположенной в секции СЕК2, в качестве регистров базы секции СЕК1 определены регистры 4 и 6. Регистр 6 будет использоваться для базирования адреса литерала в команде LIT. Значит, регистр базы в этом случае будет правильно определен и для литерала. Если бы в секции СЕК2 регистры базы для СЕК1 были уже отменены, то адрес литерала нельзя было бы базировать и оператор с именем LIT был бы ошибочным.

Комментарии в поле операндов приведенного модуля поясняют действие каждого оператора.

3.11.3. Использование фиктивных областей

Фиктивная область является средством описания областей памяти без действительного резервирования памяти. Предполагается, что память резервируется либо в другой части данного исходного модуля, либо другим исходным модулем. Фиктивная область определяется командой Ассемблера DSECT (ОПРЕДЕЛИТЬ ФИКТИВНУЮ ОБЛАСТЬ).

Оператор DSECT имеет следующий формат:

Название	Операция	Операнды
Простое символическое имя, параметр или соединение параметра с другими знаками	DSECT	Не используется

Символическое имя в поле названия является переместимым именем, значение которого равно 0, характеристика длины этого имени равна 1.

Оператор DSECT определяет начало или продолжение фиктивной области. Если в модуле встречается несколько операторов DSECT с одним и тем же именем, то первый из них определяет начало фиктивной области, а остальные определяют ее продолжение. В одном модуле может быть определено несколько фиктивных областей, но каждая из них должна быть именована.

Операторы, следующие за оператором DSECT, относятся к фиктивной области. Появление оператора CSECT, COM или DSECT с другим именем обозначает конец операторов, относящихся к данной фиктивной области.

Рассмотрим следующий пример:

Название	Операция	Операнды
CEK1	START BALR USING	X'1000' 15,0 *,15
TABL	DSECT	. . .
CEK2	CSECT	. . .
TABL	DSECT	. . .
PAB	DSECT	. . .
. . .	END	CEK1

Первый оператор DSECT с именем TABL определяет начало фиктивной области TABL. Операторы, следующие за этим оператором DSECT до оператора CSECT с именем CEK2, будут рассматриваться транслятором как операторы, описывающие фиктивную область TABL. Второй оператор DSECT с именем TABL будет определять продолжение фиктивной области TABL. Операторы, относящиеся к этой области, заканчиваются при появлении оператора DSECT с именем PAB. Этот оператор определяет вторую фиктивную область модуля—область PAB.

Структура области памяти, которая представлена в данном модуле в виде фиктивной области, описывается с помощью обычных операторов Ассемблера, которые следуют за оператором DSECT и могут иметь символические имена. Для каждой фиктивной области, определяемой в модуле, ведется счетчик адреса. Начальное значение счетчика адреса для фиктивной области всегда равно нулю, а затем увеличивается каждый раз на длину обрабатываемого оператора, принадлежащего фиктивной области. Счетчик адреса фиктивной области используется транслятором для определения значений символических имен из фиктивной области.

Команды или константы, появившиеся в фиктивной области, не образуют объектных кодов и в объектный модуль не попа-

дают, но в распечатке они будут присутствовать. Символические имена, которые именуют операторы фиктивной области, обычно используются в машинных командах и в некоторых командах Ассемблера в программных секциях модуля. Символическое имя, которое именует оператор в фиктивной области, может быть использовано в адресной константе типа А или Y только тогда, когда оно спарено с другим именем из этой же фиктивной области.

Если символические имена из фиктивной области используются в машинных командах, то в программе необходимо выполнить следующее:

записать оператор USING, информирующий транслятор о регистре базы для неявных адресов из фиктивной области и о значении базового адреса;

загрузить с помощью машинных команд указанный регистр базы действительным адресом той области памяти, которая описана фиктивной областью.

Таким образом, транслятор представит имена из фиктивной области (неявные адреса) в виде регистра базы и смещения относительно некоторого адреса фиктивной области, указанного оператором USING. При выполнении программы в регистр базы машинными командами будет загружен адрес области памяти, которая описывается в программе как фиктивная область. Следовательно, все машинные команды, которые используют имена, определяемые в фиктивной области, во время выполнения будут обращаться к адресам действительной области памяти. Например, рассмотрим следующий модуль:

Название	Операция	Операнды
PROG	CSECT	15,0
	BALR	*.15
	USING	NAME,3
	USING	2,100
	LA	4,NAME
	L	2,4
	A	2,NAME1
	ST	14
	SVC	
	DSECT	
	DS	F
	DS	F
PAB NAME NAME1	END	

В модуле определена фиктивная область, описывающая область памяти, зарезервированную в другом модуле, который Редактором будет объединен с данным в одну выполняемую программу. В программной секции PROG приведенного модуля используются имена операторов из фиктивной области (NAME, NAME1). Поэтому в модуле присутствует оператор USING с име-

нем NAME из фиктивной области, указывающий транслятору, что регистр базы 3 содержит базовый адрес из фиктивной области PAB. Имена NAME и NAME1 будут базироваться регистром 3. Смещение будет вычислено относительно начального адреса фиктивной области, так как имя NAME, указанное в операторе USING, имеет значение, равное начальному адресу фиктивной области. Перед обращением к этому модулю должна быть выполнена загрузка регистра 3 начальным адресом действительной области памяти, которая описывается с помощью фиктивной области PAB.

Предполагается, что это выполняется в другом модуле, в котором определяется действительная область и который будет объединен с данным при редактировании. Тогда при выполнении машинных команд, использующих имена фиктивной области, адреса операндов, получаемые в результате сложения содержимого регистра базы 3 и смещения, будут адресами полей действительной области памяти, описанной фиктивной областью PAB.

Рассмотрим еще один пример использования фиктивной области. Допустим, в памяти имеются три массива данных. Каждый массив данных описывается некоторой таблицей информации, структура которой для всех массивов одинакова. Каждая таблица содержит следующую информацию: идентификатор массива (8 байт); длина массива (4 байта); начальный адрес массива (4 байта).

Необходимо выполнить некоторую одинаковую обработку всех трех массивов данных. Поставленную задачу можно реализовать, написав следующую программу:

Название	Операция	Операнды
OCH	START	X'2000' основная программа
	BALR	15,0 загрузка и определение регистра
	USING	*,15 базы для основной программы
*	L	5,ADR загрузка адреса подпрограммы, за-
*		грузка регистра базы для подпро-
	L	3,ADR1 загрузка регистра базы DSECT
	BALR	14,5 14 — регистр возврата
	L	3,ADR2 загрузка регистра базы DSECT
	BALR	14,5
	L	3,ADR3 загрузка регистра базы DSECT
	BALR	14,5
	SVC	14
ADR	DC	A(OBRAB)
ADR1	DC	A(TABL1)
ADR2	DC	A(TABL2)
ADR3	DC	A(TABL3)
OBRAB	CSECT	
	USING	*,5 подпрограмма обработки
*		определение регистра базы подпро-
	USING	TABL,3 граммы
	L	10,AMAC определение регистра базы DSECT

Название	Операция	Операнды
• • •	BR	14 • • •
TABL	DSECT	описание таблицы
NAME	DS	CL8 имя массива
LMAC	DS	CL4 длина массива
AMAC	DS	CL4 адрес массива
TABL1	CSECT	
	DC	CL8 'массив 1'
	DC	F'80'
	DC	A(MAC1)
TABL2	CSECT	
	DC	CL8 'массив 2'
	DC	F'120'
	DC	A(MAC2)
TABL3	CSECT	
	DC	CL8 'массив 3'
	DC	F'320'
	DC	A(MAC3)
MAC	CSECT	массивы
MAC1	DC	X'123456'
MAC2	DC	• • •
MAC3	DC	• • •
• • •	END	OCH • • •

В приведенной программе выделена подпрограмма обработки массива. В этой подпрограмме оператором DSECT с именем TABL определена фиктивная область, описывающая структуру таблицы информации о каждом массиве. При выполнении обработки в подпрограмме используются символические имена из фиктивной области. Для базирования адресов из фиктивной области определен регистр базы 3. Неявные адреса из фиктивной области TABL (например, AMAC) будут базироваться регистром 3. Подпрограмма обработки выделена в отдельную программную секцию с именем OBRAV. Для нее в качестве регистра базы определен регистр 5. Загрузка регистра базы для подпрограммы выполняется в основной программе. Программная секция с именем OCH, определяемая оператором START, представляет собой основную программу. Здесь выполняется обращение к подпрограмме для обработки каждого массива. Для обращения к подпрограмме используется команда BALR, которая выполняет переход по адресу, находящемуся в регистре 5. В регистр 5 загружается адрес подпрограммы. Адрес возврата, который используется для выхода из подпрограммы, сохраняется в регистре 14. Загрузка адреса подпрограммы в регистр 5 одновременно является и загрузкой регистра базы для секции OBRAV, так как регистр 5 указан оператором USING регистром базы для секции OBRAV.

Обращение к подпрограмме обработки массива выполняется три раза. Перед каждым обращением выполняется загрузка регистра 3, являющегося регистром базы для имен из фиктивной области. При выполнении подпрограммы в регистре 3 будет находиться начальный адрес таблицы информации обрабатываемого массива. Адреса операндов в командах подпрограммы, использующих имена из фиктивной секции, будут получены сложением содержимого регистра базы 3 и смещения. Смещение для элементов каждой таблицы относительно начала таблицы и смещение элементов относительно начала фиктивной области одинаково, в регистре базы 3 при выполнении будет адрес действительной таблицы. Поэтому при выполнении подпрограммы будут выполняться действия над содержимым той таблицы (соответственно и над массивом данных), адрес которой будет загружен в регистр базы 3.

3.11.4. Использование общих областей

При написании программы в виде отдельных исходных модулей программист может осуществлять связь между разными модулями с помощью общей области.

Общая область представляет собой неименованную область памяти, к которой могут обращаться многие независимые модули, впоследствии соединенные и загруженные для выполнения как одна программа. Если фиктивная область описывает область памяти, которая обязательно где-то зарезервирована в программе (в другой части модуля или в другом модуле), то для общей области место в памяти ни одним из отдельно транслируемых модулей программы не определяется. В этих модулях может только определяться наличие общей области и описываться ее структура, а ее местоположение в основной памяти определяется при редактировании.

Для определения общей области предназначена команда Ассемблера **COM** (**ОПРЕДЕЛИТЬ ОБЩУЮ ОБЛАСТЬ**).

Оператор **COM** имеет следующий формат:

Название	Операция	Операнды
Символическое имя перехода или пробел	COM	Не используется

В исходном модуле может быть определена только одна неименованная общая область. Общая область может прерываться операторами **CSECT** или **DSECT**, в этом случае в модуле может появиться несколько операторов **COM**. Первый из них указывает начало общей области, остальные—продолжение. Операторы, следующие за оператором **COM**, будут отнесены к операторам, описывающим общую область.

Когда Редактор создает выполняемую программу, он резервирует для общей области область памяти, которая предшествует всей программе. Если Редактор объединяет несколько модулей, каждый из которых определяет общую область, то объем резервируемой памяти равен наибольшей общей области. Общая область всегда начинается на границе двойного слова. Начальный адрес выполняемой программы определяется Редактором с учетом наличия и длины общей области.

Общая область в каждом исходном модуле может быть разбита на поля с помощью любых операторов языка Ассемблера. Поля могут присваиваться имена.

Имена полей используются в программных секциях модуля для обращения к общей области. Если символические имена из общей области используются в машинных командах (неявные адреса), то в модуле должен быть определен регистр базы, доступный для этих имен. Доступным регистром базы для имен из общей области будет тот из регистров, определенных операторами USING, у которого базовый адрес принадлежит к общей области. Этот базовый адрес должен задаваться именем некоторого оператора из общей области, потому что оператор COM не имеет имени. При трансляции назначение адресов элементам общей области начинается с нуля.

В приведенном ниже примере выполняется определение и загрузка регистра базы для имен из общей области.

Название	Операция	Операнды
...	L	I,=A(NAME)
	USING	NAME,I
	MVC	OB(16),=4C'ABCD'
...	COM	общая область
NAME	DS	16F
OB	DS	16C
KON	DC	F'400'
PROG	CSECT	
...	END	...

Поля (элементы) из общей области будут адресоваться относительно оператора с именем NAME, потому что имя NAME указано транслятору в качестве базового адреса оператором USING. Общий регистр I будет доступным регистром базы для неявного адреса OB, используемого в команде MVC: базовый адрес и адрес OB оба из общей области, адрес OB больше базового адреса NAME, и разность между ними не превышает 4095 (в общей области имени NAME будет присвоено значение нуля, а имени OB—64). Транслятор представит неявный адрес OB в виде регистра базы I и смещения, равного 64. При выполнении программы

значение имени NAME, относительно которого вычислял смещение транслятор, будет загружено в регистр 1. Этим будет обеспечена правильная установка действительного адреса операнда для команды MVC.

Команда и константы, появляющиеся в общей области (например, KON в приводимом примере), не образует объектного кода и в объектный модуль не попадают, но в распечатке они будут присутствовать. Данные могут быть помещены в общую область только во время выполнения программы.

Если определение элементов общей области сделано одинаковым образом в каждом модуле программы, то при ссылке к общей области из любого модуля будет выполняться обращение к одним и тем же элементам. Однако в разных модулях один и тот же элемент общей области может именоваться по-разному.

Допустим, что выполняемая программа, кроме приводимого ранее модуля, будет содержать еще один модуль. Второй модуль использует результат вычисления первого модуля, который сохраняется в общей области, начиная с байта 64.

Общая область второго модуля может быть описана следующим образом:

Название	Операция	Операнды
...	L	1, = A(A)
	USING	A, 1
	L	3, C
...	COM	...
A	DS	8F
B	DS	8F
C	DS	16C
D	DS	F
...	END	...

При выполнении первого модуля программы в общую область помещаются данные. Во втором модуле именно эти данные используются, потому что оба модуля обращаются к одной и той же области памяти: начиная с байта 64 общей области. То, что общая область описана в модулях по-разному и символические имена полей общей области в модулях разные, не влияет на содержимое общей области.

3.11.5. Символическая связь между исходными модулями

Программа на языке Ассемблера может состоять из нескольких исходных модулей, которые могут транслироваться отдельно.

Информация, указываемой операторами CSECT, START, DSECT, COM, не всегда достаточно для осуществления связи между секциями, например, в случае, когда необходимо выполнить переход в другой модуль, причем не в начало секции. Символическую связь на языке Ассемблера в таких случаях можно осуществить с помощью специальных символических имен связи и констант типа V, о которых транслятор также выдает информацию Редактору. С помощью этой информации Редактор устанавливает действительные адреса символических имен.

Символические имена связи—это такие имена, которые могут быть определены в одном модуле и использоваться в другом. Существуют два типа символических имен связи: входные и внешние.

Если символическое имя, используемое в данном модуле, определено (т. е. используется в поле названия) в некотором другом модуле, то оно называется внешним именем. Если символическое имя определено в данном модуле, но может использоваться в другом модуле (например, для перехода или ссылки к данным), то оно называется входным именем.

Для каждого исходного модуля программы программист должен знать, какие символические имена у него будут входными и какие внешними. Он должен в каждом модуле сообщить это транслятору, так как транслятор должен подготовить информацию для Редактора. Программист сообщает транслятору об именах связи с помощью команд Ассемблера ENTRY, EXTRN и констант типа V. Транслятор передает эту информацию Редактору, размещая ее в формируемый для каждого модуля словарь внешних имен.

Команда Ассемблера ENTRY (ОПРЕДЕЛИТЬ ВХОДНОЕ ИМЯ) называет символические имена, которые определяются в данном модуле, но могут быть использованы другими, отдельно транслируемыми модулями программы.

Оператор ENTRY имеет следующий формат:

Название	Операция	Операнды
Символическое имя перехода или пробел	ENTRY	Одно или несколько переместимых символических имен, разделенных запятыми

Символические имена, указанные в поле операндов оператора ENTRY, должны появиться в данном модуле как названия операторов. Максимально модуль может содержать 100 имен ENTRY. В поле операндов оператора ENTRY нельзя указывать имя, определенное в неименованной программной секции, в фиктивной или общей области. Если другой модуль использует имя программной секции, то его можно не идентифицировать как входное имя опе-

ратором ENTRY. Транслятор всегда помещает информацию об именах программных секций в словарь внешних имен.

Команда Ассемблера EXTRN (ОПРЕДЕЛИТЬ ВНЕШНЕЕ ИМЯ) называет символические имена, которые используются данным модулем, но определяются в другом модуле. Каждое внешнее имя, необходимое в данном модуле, должно быть названо в операторе EXTRN. Это относится и к именам, называющим программные секции.

Оператор EXTRN имеет следующий формат:

Название	Операция	Операнды
Символическое имя перехода или пробел	EXTRN	Одно или несколько переместимых символических имен, разделенных запятыми

Имена из поля операндов оператора EXTRN не должны появляться как названия операторов в данном модуле. Так как символическое имя из поля операндов оператора EXTRN определяется в другом модуле, то транслятор присваивает ему характеристику длины, равную единице, и нулевое значение. Каждое внешнее имя рассматривается как имя, имеющее собственную переместимость, т. е. каждое имя относится к отдельной программной секции. Поэтому внешние имена в простых переместимых выражениях не могут быть спарены.

Внешнее имя можно указать в адресной константе типа V. В этом случае это внешнее имя не нужно определять оператором EXTRN. Обычно адресная константа типа V используется для организации внешних переходов. Символическое внешнее имя, использованное в адресной константе типа V, не должно использоваться как операнд в других операторах Ассемблера.

Информация о внешних именах, указанных операторами EXTRN и константами типа V, помещается транслятором в словарь внешних имен. Если одно и то же имя появляется во внешней адресной константе типа V и в поле названия операторов DSECT или CSECT, то оно рассматривается как разные символические имена.

Общее число программных секций и фиктивных областей плюс общая область и число внешних имен в операторах EXTRN и в адресных константах типа V в модуле не должно превышать 255.

Рассмотрим несколько примеров символического соединения исходных модулей.

Допустим, необходимо организовать переход из одной программной секции в другую секцию, когда секции транслируются отдельно. Это можно организовать, например, следующим способом:

создать в первой программной секции внешнюю адресную константу типа V с соответствующим внешним именем (из другой секции);

загрузить полученную константу в общий регистр и перейти к другой программной секции, используя этот регистр.

Приведенный ниже исходный модуль показывает, как можно организовать переход в программную секцию другого исходного модуля, названную OBRAB.

Название	Операция	Операнды
OCHPROG BEGIN	CSECT BALR USING	15,0 *,15
. . .	L . . .	3,VIMA
VIMA	BR DC END	3 V(OBRAB) BEGIN

Внешняя адресная константа типа V, названная именем VIMA, указывает внешнее имя OBRAB в другом исходном модуле, транслируемом отдельно. После трансляции значение этой константы равно нулю. Во время редактирования, когда будет выполняться объединение двух отдельно транслируемых модулей в одну выполняемую программу, будет установлено действительное значение этой константы. При выполнении программы установленное уже действительное значение этой константы будет загружаться в регистр 3, а затем будет выполнен переход по адресу, загруженному в этот регистр, т. е. по адресу OBRAB в другой секции. Как отмечалось ранее, внешнее имя OBRAB является именем программной секции, т. е. появляется в поле названия оператора CSECT. В этом случае имя OBRAB может не идентифицироваться оператором ENTRY в том исходном модуле, который содержит эту секцию. Если бы это имя называло, допустим, некоторую машинную команду, то в исходном модуле, содержащем эту команду, должен был бы присутствовать следующий оператор:

Название	Операция	Операнды
	ENTRY	OBRAB

Переход в другой модуль можно было организовать и следующим способом:

Название	Операция	Операнды
OCHPROG	CSECT	
BEGIN	EXTRN	OBAB
	BALR	15,0
	USING	*,15
...		
	L	3,VIMA
	BR	3
VIMA	DC	A(OBRAB)
...		
	END	BEGIN

В этом случае подготовка константы с адресом перехода выполняется с помощью двух операторов (EXTRN, DC), а в первом случае выполнялась с помощью одного оператора (DC). Поэтому чаще всего для организации перехода в другой модуль используется первый способ. Переход обычно выполняется по адресу в регистре, а не используется неявный адрес в команде безусловного перехода, так как в последнем случае необходимо добавлять еще команды определения регистра базы для внешнего имени.

Предположим, что в исходном модуле необходимо использовать данное из другого исходного модуля, транслируемого отдельно, в котором имя данного идентифицировано как входное оператором ENTRY. Это можно сделать следующим образом:

идентифицировать имя данного оператором EXTRN и создать адресную константу из этого символического имени;

загрузить полученную константу в общий регистр и использовать этот регистр как регистр базы.

Например, для использования данных из области DATE, которая находится в другом исходном модуле, можно использовать следующий исходный модуль:

Название	Операция	Операнды
OCHPROG	CSECT	
BEGIN	BALR	15,0
	USING	*,15
...		
	EXTRN	DATE
...		
	L	4,BASD
	USING	DATE,4
	A	3,DATE
BASD	DC	A(DATE)
	END	BEGIN

Имя DATE в модуле определено как внешнее имя оператором EXTRN. Это имя используется в команде A (СЛОЖЕНИЕ). Но оно представляет собой неявный адрес, который должен быть представлен транслятором в виде регистра базы и смещения. Для имени DATE будет доступным только тот регистр базы, который содержит базовый адрес из той же секции, что и имя DATE. Но так как каждое внешнее имя в исходном модуле относится транслятором к отдельной секции, то при использовании внешних имен в поле операндов машинных команд необходимо в модуле записать оператор USING отдельно для каждого используемого внешнего имени. Оператор USING может определить базовый регистр только для одного внешнего имени, потому что в данном модуле больше нет символических имен с таким же признаком переместимости.

В приведенном модуле для внешнего имени DATE в качестве регистра базы определяется общий регистр 4 (в операторе USING, определяющем регистр 4 регистром базы, содержится имя DATE, т. е. указывается базовый адрес из той же секции, что и неявный адрес DATE). Транслятор, используя указание оператора USING, представит неявный адрес DATE в виде регистра базы и смещения. Но необходимо еще позаботиться о загрузке регистра 4 базовым адресом, указанным транслятору в операторе USING. Для этого создана адресная константа и присутствует команда загрузки этой адресной константы в регистр 4. Значение адресной константы будет установлено при редактировании, а во время выполнения в регистр 4 будет загружаться необходимое значение адреса из другого модуля.

МАКРОСРЕДСТВА

4.1. ВОЗМОЖНОСТИ МАКРОСРЕДСТВ

Рассмотренные ранее операторы языка позволяют записывать решение задачи в виде набора машинных команд и команд Ассемблера. Каждая машинная команда выполняет только одну элементарную операцию (например, сложение двух чисел), поэтому решение даже несложной задачи реализуется набором многочисленных команд, число которых может достигать тысячи. При этом в программе могут повторяться одинаковые наборы команд. Макросредства позволяют назвать наборы повторяющихся команд некоторым именем и в программе не перечислять все эти команды, а только записать команду, которой именуется данный набор. Например, в следующем примере часто встречаются две команды: L и ST.

Название	Операция	Операнды
	L . .	R,AREA
	ST	R,WORKA
	L . .	R,AREA
	ST	R,WORKA
	L . .	R,AREA
	ST	R,WORKA

Эти команды можно объединить в набор и назвать именем REST. Тогда вместо двух команд можно записать один оператор, называемый макрокомандой. Код операции макрокоманды должен совпадать с названием набора команд, который соответствует данной макрокоманде, и не должен совпадать с кодом операций операторов Ассемблера. В результате программа примет вид:

Название	Операция	Операнды
	REST	
	REST	
	REST	
	. . .	

Вместо каждой макрокоманды REST в исходную программу будут вставляться команды L и ST. Чтобы транслятор мог обработать макрокоманды, используемые в исходной программе, макросредства позволяют группу команд, которые необходимо вставить по макрокоманде в исходную программу, оформить специальным образом в макроопределение. Макроопределение записывается в самом начале программы. Так, если в приведенном примере машинные команды L и ST оформить в виде макроопределения, то для всех макрокоманд REST транслятор вставит команды макроопределения, и программа примет первоначальный вид.

Рассмотрим следующий пример.

Название	Операция	Операнды
	L . .	R1,AREA1
	ST	R1,AREA2
	L . .	R2,AREA3
	ST	R2,AREA4
	L . .	R3,AREA5
	ST	R3,AREA6
	. . .	

От предыдущего примера он отличается тем, что в повторяющихся операторах записаны разные операнды. Но макросредства позволяют и такие операторы оформлять в виде макроопределения. В этом случае изменяющиеся части операторов записываются с помощью параметров, т. е. таких символических имен, которые во время генерации заменяются нужными значениями.

Параметр записывается как знак &, за которым следует от одной до семи букв или цифр; первой за знаком & должна быть буква. Теперь в макроопределении будут записаны следующие операторы:

Название	Операция	Операнды
	L	&R,&P1
	ST	&R,&P2

В макроопределении используются параметры &R, &P1 и &P2, которые заменяются теми значениями, которые указываются в макрокоманде. В результате пример будет выглядеть следующим образом.

Имя	Операция	Операнды
	REST	R1, AREA1, AREA2
	REST	R2, AREA3, AREA4
	REST	R3, AREA5, AREA6
	...	

В первой макрокоманде параметру &R соответствует значение R1 и, когда транслятор вставляет команды макроопределения, параметр &R заменяется значением R1. Аналогично параметр &P1 заменяется значением AREA1, а параметр &P2 — значением AREA2. Поэтому вместо первой макрокоманды REST будут записаны такие команды:

```
L R1, AREA1
ST R1, AREA2
```

В следующей макрокоманде указаны другие значения параметров: R2, AREA3, AREA4. Параметры &R, &P1 и &P2 заменяются новыми значениями и вместо второй макрокоманды REST вставляются следующие команды:

```
L R2, AREA3
ST R2, AREA4
```

А так как в третьей макрокоманде также указаны новые значения параметров, то вместо нее появятся команды:

```
L R3, AREA5
ST R3, AREA6
```

Таким образом, если составить одно макроопределение, а в макрокоманде указать разные значения параметров, то из одного макроопределения будут вставляться разные наборы команд, которые назовем макрорасширениями. Процесс построения макрорасширения будем называть генерацией.

Рассмотренные примеры не охватывают всех возможностей, предоставляемых макросредствами, они позволяют уяснить лишь общие понятия.

4.2. МАКРООПРЕДЕЛЕНИЕ

Прежде чем использовать в программе макрокоманду, необходимо иметь макроопределение, которое соответствует данной макрокоманде. Макроопределение содержит последовательность операторов, которую должен обработать транслятор при появлении макрокоманды в исходной программе. Макроопределение может содержаться в том исходном модуле, в котором оно используется, или в подбиблиотеке Ассемблера библиотеки исходных модулей.

Если макроопределение записывается в программе, то оно должно стоять перед всеми операторами программы, за исключением следующих: ICTL, ISEQ, TITLE, PRINT, EJECT, SPACE и комментариев. Эти операторы (кроме ICTL) можно записывать между макроопределениями.

Если макроопределение должно использоваться в нескольких программах, его помещают в подбиблиотеку Ассемблера. В этом случае его не нужно записывать в исходной программе. В подбиблиотеку Ассемблера, кроме макроопределений, включаются также части программы, которые написаны на языке Ассемблера и вызываются оператором COPY.

4.2.1. Состав макроопределения

Любое макроопределение должно состоять из следующих операторов:

- оператора MACRO, который указывает начало макроопределения;

- оператора прототипа (прототип), который определяет формат макрокоманды, соответствующей данному макроопределению;

- набора операторов, составляющих макроопределение. Этими операторами могут быть любые операторы языка Ассемблера, кроме END, ICTL, ISEQ, PRINT, MACRO и MEND;

- оператора MEND, указывающего конец макроопределения.

В состав макроопределения могут входить макрокоманды. Их называют внутренними макрокомандами. Макроопределение внутренней макрокоманды может содержать произвольное количество других макрокоманд.

Формат оператора MACRO:

Название	Операция	Операнды
Пробел	MACRO	Не используется

Оператор MACRO только указывает начало макроопределения, никаких других функций он не выполняет.

Оператор MEND определяет конец макроопределения. Его формат:

Название	Операция	Операнды
Пробел	MEND	Не используется

Рассмотрим пример макроопределения.

Название	Операция	Операнды	Идентификация
	MACRO		1
	REST	&R,&P1,&P2	2
	L	&R,&P1	3
	ST	&R,&P2	4
	MEND		5

Оператор 1 (MACRO) и оператор 5 (MEND) указывают границы макроопределения. Оператор 2 (REST), записываемый сразу за оператором MACRO, является прототипом. Из операторов 3 и 4 формируется макрорасширение.

4.2.2. Оператор прототипа

Оператор прототипа всегда должен записываться вторым оператором макроопределения, сразу за оператором MACRO. Оператор прототипа определяет название макроопределения (это название является кодом операции макрокоманды, соответствующей макроопределению) и названия параметров, которые можно использовать в данном макроопределении.

В зависимости от того, как записаны операнды прототипа, различаются позиционные, ключевые и смешанные макроопределения. Позиционный прототип имеет следующий формат:

Название	Операция	Операнды
Параметр или про- бел	Простое имя (код операции макрокоманды)	Параметры, разделен- ные запятыми

Параметры, которые записываются в поле названия и в поле операндов прототипа, называются постоянными.

Постоянные параметры можно использовать в любом поле любого оператора макроопределения. При обработке макроопределения по макрокоманде постоянные параметры будут заменяться теми значениями, которые соответствуют им в макрокоманде. Значения параметров в макрокоманде должны записываться в том же порядке, в котором перечислены параметры в прототипе. Постоянному параметру из поля названия прототипа соответствует значение, записанное в поле названия макрокоманды. Макроопределение с позиционным прототипом называется позиционным. Позиционному макроопределению должна соответствовать позиционная макрокоманда.

Ключевой прототип отличается от позиционного прототипа правилами записи операндов. Каждый операнд ключевого прототипа должен состоять из постоянного параметра, знака равенства и, если нужно, стандартного значения параметра. В качестве

последнего может быть записано все то, что можно использовать как операнд макрокоманды, кроме параметра (правила записи операндов макрокоманды приведены в разд. 4.3.). Макроопределение с ключевым прототипом называется ключевым. Ключевому макроопределению должна соответствовать ключевая макрокоманда.

Примеры правильных операндов ключевых прототипов:

&PI=
&READ=X'I'

Примеры неправильных операндов ключевого прототипа:

CARD=7 (нет постоянного параметра);
&TYP (нет знака равенства);
&TWO=5 (непосредственно за постоянным параметром не следует знак равенства).

В следующем ключевом прототипе указаны четыре ключевых операнда. В поле названия прототипа записан постоянный параметр. Мнемонический код операции прототипа MOVE.

Название	Операция	Операнды
&N	MOVE	&R=2,&A=F1,&T=X'A',&F=

Часть операндов смешанного прототипа записывается как позиционные операнды, остальные — как ключевые. Все позиционные операнды должны предшествовать ключевым операндам. Макроопределение со смешанным прототипом называется смешанным. Смешанному макроопределению должна соответствовать смешанная макрокоманда. В следующем прототипе записаны три позиционных операнда и два ключевых:

Название	Операция	Операнды
&N	MOVE	&A,&B,&C,&D=5,&K=L1

Оператор прототипа можно записывать на бланке кодирования по правилам, отличным от правил записи других операторов. Оператор прототипа может иметь столько строк продолжения, сколько необходимо для его записи. На каждой строке можно записывать операнды и комментарии: полностью записать очередной операнд, за ним поставить запятую, а через один или более пробелов можно давать комментарий. В этом случае с колонки продолжения следующей строки нужно начинать запись следующего операнда. Если за операндом стоит не запятая, а пробел, то это значит, что данный операнд последний.

В приведенном ниже примере комментарии записаны в первой и третьей строках оператора.

Название	Операция	Операнды	
	REST	&R, THIS IS A REGISTER &P, &P1 WORK AREAS	X X

Этот же прототип можно записать следующим образом:

Название	Операция	Операнды
	REST	&R, &P, &P1

4.2.3. Соединение параметров с другими знаками

Иногда при составлении макроопределения требуется соединить параметр с другим параметром или с набором каких-нибудь знаков. Такое соединение выполняется по следующим правилам. Если нужно соединить некоторые знаки с параметром, то параметр записывается сразу же за этими знаками без разделителя. Если нужно соединить параметр с другими знаками, то за параметром записывается точка, а за ней те знаки, с которыми соединяется параметр. Точка не обязательна, если соединяются два параметра или параметр со знаками, начинающимися со специального знака (кроме левой скобки и точки). Рассмотрим пример.

Название	Операция	Операнды	Идентификация
* макроопределение			Ø
	MACRO		1
	EX1	&P1, &P2, &P3, &P4	2
	L&P1	2, &P2.A	3
	S&P1	2, &P2+C	4
	ST&P1	2, &P3. (&P4)	5
	MEND		6
* макрокоманда и макрорасширение			
	EX1	H, POOL, RESULT, 4	
	LH	2, POOLA	
	SH	2, POOL+C	
	STH	2, RESULT(4)	

В этом примере приведено макроопределение, соответствующая ему макрокоманда и макрорасширение, сгенерированное по макрокоманде. В качестве идентификации операторов макропре-

деления записаны номера операторов. Они не появляются в операторах макрорасширения. Эти номера можно было бы записать как комментарию в поле операндов операторов макроопределения через один или несколько пробелов после операндов. В этом случае они появились бы и в операторах, которые попадают в макрорасширение, причем всегда через один пробел после операндов.

Параметр &P1 используется для изменения кода операции. С этим параметром соединяются знаки L, S, ST, которые записаны перед параметром &P1, поэтому точку перед параметром записывать нельзя. Так как параметру &P1 в макрокоманде соответствует знак H, то он соединяется с предыдущими знаками и в результате генерируются мнемонические коды операций машинных команд.

В операторе 3 параметр &P2 соединяется со знаком, стоящим за ним, поэтому после параметра записывается точка, но в сгенерированном операторе ее не будет.

В операторах 4 и 5 параметры &P2 и &P3 соединяются со знаками, начинающимися с разделителя (знаки «+» и «(»). Разделитель — это знак, отличный от букв и цифр. В операторе 4 точку после параметра &P2 записывать не обязательно, но в операторе 5 параметр &P3 соединяется со знаком «(», следовательно, точка обязательна. В операторе 5 параметр &P4 соединяется со знаком «)», в этом случае точку записывать не обязательно. Данные правила применимы для соединения параметров во всех операторах языка Ассемблера, в макроопределениях и в основной части программы (вне макроопределений).

4.3. МАКРОКОМАНДА

Макрокоманда содержит всю информацию, необходимую для генерации макрорасширения из соответствующего макроопределения. Формат макрокоманды:

Название	Операция	Операнды
Любое символическое имя или пробел	Код операции	Операнды, разделенные запятыми

В поле названия макрокоманды может присутствовать имя, которое является значением параметра, записанного в поле названия прототипа. Мнемонический код операции у макрокоманды должен быть таким же, как и у прототипа соответствующего макроопределения.

Правила записи макрокоманды совпадают с правилами записи оператора прототипа. Для записи макрокоманды можно использовать любое количество строк продолжения.

4.3.1. Операнды макрокоманды

В позиционной макрокоманде операнды записываются в том порядке, в котором они соответствуют постоянным параметрам, т. е. первым записывается операнд, соответствующий первому параметру прототипа, вторым — операнд, соответствующий второму параметру, и т. д.

Иногда нет необходимости указывать в макрокоманде значение некоторого постоянного параметра. Такой операнд считается опущенным. В этом случае вместо операнда записывается запятая, которая должна была отделить данный операнд от следующего. Если опускается подряд несколько операндов, то записывается столько же запятых, причем если они последние в макрокоманде, то запятые можно не писать.

В следующем примере приведены прототип и соответствующая макрокоманда, в которой опущены второй, третий, шестой и седьмой операнды.

Название	Операция	Операнды
	PR PR	&P1,&P2,&P3,&P4,&P5,&P6,&P7 POOL,,,128,'TYPE'

Во время генерации постоянные параметры, которым соответствуют опущенные операнды макрокоманды, опускаются. Это правило используется в следующем примере.

Название	Операция	Операнды	Идентификация
* макроопределение	MACRO		1
	REST	&P1,&P2,&P3	2
	L&P1	2,&P2	3
	S&P1	2,&P3	4
	MEND		5
* макрокоманда и макрорасширение	REST	,AREA1,AREA2	
	L	2,AREA1	
	S	2,AREA2	

В макрокоманде первый операнд, соответствующий параметру &P1, опущен. Постоянный параметр &P1 присутствует в поле операции оператора 3 макроопределения. По данной макрокоманде для оператора 3 генерируется код операции L. Необходимо учитывать, что опущенному операнду соответствует не пробел, а так называемое пустое знаковое значение, которое определяет отсутствие знаков. Параметр при его замене пустым знаковым значением опускается.

В ключевом прототипе перечисляются постоянные параметры и указываются их стандартные значения. В ключевой макрокоманде

де в качестве операндов записываются только те операнды, которые изменяют стандартные значения параметров. Остальные параметры в макрокоманде можно не перечислять. Каждый операнд ключевой макрокоманды состоит из ключевого слова, непосредственно за которым следует знак равенства и значение параметра. Ключевое слово должно соответствовать одному из постоянных параметров, перечисленных в поле операндов ключевого прототипа. Ключевое слово соответствует параметру, если знаки ключевого слова идентичны знакам постоянного параметра, следующим за знаком &.

Примеры правильных операндов ключевой макрокоманды:

S4=F'5'
TO=

Примеры неправильных операндов ключевой макрокоманды:

&D=4 (ключевое слово начинается со знака &);
P1 (нет знака равенства);
K0=2 (непосредственно за ключевым словом не следует знак равенства).

Операнды в ключевой макрокоманде могут быть записаны в любом порядке. Замена постоянных параметров в операторах ключевого макроопределения выполняется по следующим правилам:

а) если постоянный параметр указывается в поле операндов прототипа и макрокоманда содержит ключевое слово, которое соответствует постоянному параметру, то значение, присвоенное ключевому слову, замещает постоянный параметр. Этим значением может быть и опущенный операнд;

б) если макрокоманда не содержит ключевого слова, которое соответствует постоянному параметру из поля операндов прототипа, то постоянный параметр заменяется стандартным значением, присвоенным постоянному параметру оператором прототипа. Этим значением может быть и опущенный операнд.

Эти правила используются при обработке следующего макроопределения:

Название	Операция	Операнды	Идентификация
* макроопределение			
&N &N	MACRO		1
	MOVE	&R=2,&A=S,&T=,&F=	2
	ST	&R,&A	3
	L	&R,&F	4
	ST	&R,&T	5
	L	&R,&A	6
	MEND		7
* макрокоманда и макрорасширение			
NAM	MOVE	T=FA,F=FB,A=F1	8
NAM	ST	2,F1	9
	L	2,FB	
	ST	2,FA	
	L	2,F1	

Прототип (оператор 2) присваивает значения 2 и S соответственно параметрам &R и &A. Макрокоманда (оператор 9) присваивает значения FA, FB и F1 соответственно ключевым словам T, F и A. В поле названия макрокоманды записано имя NAM, а в поле названия прототипа — параметр &N. Последний записан также и в поле названия оператора 3, поэтому этот параметр в поле названия оператора 3 заменяется значением NAM.

Параметр &R во всех операторах макроопределения заменяется значением 2, которое присвоено ему оператором прототипа (ключевое слово R в макрокоманде отсутствует).

Смешанному макроопределению должна соответствовать смешанная макрокоманда. Часть ее операндов записывается по правилам записи операндов позиционной макрокоманды, а остальные — по правилам записи операндов ключевой макрокоманды. Все позиционные операнды смешанной макрокоманды должны предшествовать ее ключевым операндам. В дальнейшем будут рассматриваться только позиционные макроопределения и макрокоманды.

Операндом макрокоманды может быть комбинация любых знаков кода ДКОИ. При записи операндов необходимо соблюдать некоторые правила относительно апострофов, скобок, знаков равенства, знаков &, запятых и пробелов.

Операнд может состоять из последовательности знаков, начинающейся и оканчивающейся апострофом. Оба апострофа входят в значение операнда и называются парными. Внутри последовательности может находиться четное число подряд стоящих апострофов, но они не считаются парными. Если внутри парных апострофов встречается запятая, то она не считается разделителем операндов. Внутри парных апострофов может быть пробел, который входит в состав операнда и не является разделителем. В следующем примере первый и второй, третий и шестой апострофы являются парными. Запятая и пробел внутри них не считаются разделителями.

'A'B'C'DOE'

В операнде макрокоманды можно использовать круглые скобки, при этом число правых скобок должно быть равным числу левых. Все скобки входят в значение операнда. Скобки между парными апострофами не рассматриваются при определении парных скобок. Запятая внутри парных скобок не считается разделителем операндов, например,

(.A)B(C(E)'F)D)

В этом примере первая и вторая, третья и седьмая, четвертая и шестая скобки парные. Запятая не является разделителем.

Знак равенства можно использовать в операнде макрокоманды. В этом случае он должен быть либо первым знаком операнда,

либо стоять между парными апострофами или парными скобками, как, например, в следующих операндах:

=C'A1'
'ONE=1'
A(B=6)

Каждый знак & в операнде макрокоманды должен записываться двумя знаками &. В операнде макрокоманды может присутствовать параметр. Перед обработкой макрокоманды он заменяется соответствующими ему знаками. Например, если в операнде макрокоманды AB&C12 параметру &C12 соответствует значение 1+X'1', то фактическим операндом макрокоманды является значение AB1+X'1'.

4.3.2. Список операндов

В качестве операнда макрокоманды может использоваться список. Список представляет собой один или несколько операндов, разделенных между собой запятыми и заключенных в круглые скобки. Опущенные элементы списка обрабатываются так же, как опущенные операнды. В виде списка обычно представляются операнды, одинаковые по назначению.

Если список содержит n операндов, а этот список соответствует параметру &P, то к элементу m списка можно обратиться с помощью записи &P(m). Если параметру &P в макрокоманде соответствует список, а в операторе макроопределения используется параметр &P, то при генерации &P заменяется всеми знаками списка, включая парные скобки. Если же в макроопределении используется обозначение списка, но операнд макрокоманды не список, то &P(1) обозначает обращение ко всему операнду, а &P(n), где $n > 1$, — обращение к опущенному операнду.

В приведенном ниже примере параметру &PAR в макрокоманде соответствует список, который состоит из трех операндов:

Название	Операция	Операнды
	MACRO	макроопределение
	EX1	®,&PAR
	L&PAR(5)	®,&PAR(2)
	A&PAR(5)	®,&PAR(3)
	ST&PAR(5)	®,&PAR(1)
	L	1,=A&PAR
	MEND	
	EX1	3,(A1,A2,A3)

В командах макроопределения имеется обращение как к отдельным элементам списка, так и ко всему списку. В результате генерируется следующее макрорасширение:

Название	Операция	Операнды
	L	3,A2
	A	3,A3
	ST	3,A1
	L	1,=A(A1,A2,A3)

Операнд макрокоманды, который соответствует постоянному параметру &PAR, является списком. В поле операции параметр &PAR(5) указывает пятый элемент списка, который в макрокоманде считается опущенным операндом. При генерации макроопределения постоянный параметр опускается. Остальные операнды присутствуют, и эти значения заменяют параметры при генерации макроопределения. Вместо параметра &PAR подставляются все знаки, из которых состоит список.

4.4. СИСТЕМНЫЕ ПАРАМЕТРЫ

Наряду с постоянными параметрами в макроопределении могут использоваться системные параметры. Значения им присваиваются транслятором Ассемблера. Системные параметры, как и другие параметры, могут использоваться в любом поле оператором макроопределения. Существуют три системных параметра: &SYSNDX, &SYSECT и &SYSLIST.

4.4.1. Параметр &SYSNDX

Системный параметр &SYSNDX применяется для создания уникальных символических имен в операторах макроопределения при многократном его использовании в программе.

Параметр &SYSNDX — это четырехразрядный порядковый номер макрокоманды. Когда транслятор встречается в программе первую макрокоманду, параметру &SYSNDX присваивается начальное значение 0001. Для каждой следующей макрокоманды данной программы это значение увеличивается на единицу, даже если макрокоманда с таким кодом операций обрабатывалась уже ранее. Значение &SYSNDX — константа, она не изменяется при обработке макроопределения для данной макрокоманды. Если в макроопределении есть внутренняя макрокоманда, то для соответствующего ей макроопределения устанавливается свое значение параметра SYSNDX.

Для создания уникального имени этот параметр соединяется с другими знаками, максимальное количество которых не должно быть более четырех, потому что простое имя не может содержать больше восьми знаков. Первый из этих знаков должен быть буквой.

Возможность использования системного параметра &SYSNDX

показана на следующем примере. Допустим, что в программе неоднократно используется такое макроопределение:

Название	Операция	Операнды	Идентификация
LAB&SYSNDX	MACRO		1
	EX1	&P3,&P4	2
	CLC	&P3,=F'Ø'	3
	BE	LAB&SYSNDX	4
	S	3,&P3	5
	ST	3,&P4	6
	MEND		7

В этом макроопределении имя LAB&SYSNDX используется в поле названия как имя, необходимое для перехода к оператору 6.

Предположим, что макрокоманда, соответствующая данному макроопределению, является двадцать первой макрокомандой, которая обрабатывается при генерации. Тогда параметр &SYSNDX получит значение 0021, а имя будет иметь вид LAB0021. В результате сгенерируется макрорасширение:

Название	Операция	Операнды
LABØØ21	EX1	C,D макрокоманда
	CLC	C,=F'Ø'
	BE	LABØØ21
	S	3,C
	ST	3,D

Если эта же макрокоманда является сто пятой обрабатываемой макрокомандой, то &SYSNDX получит значение 0105. В этом случае сгенерируется другое макрорасширение:

Название	Операция	Операнды
LABØ1Ø5	EX1	M,N макрокоманда
	CLC	M,=F'Ø'
	BE	LABØ1Ø5
	S	3,M
	ST	3,N

В результате в программе не будет повторяющихся имен.

4.4.2. Параметр &SYSECT

Системный параметр &SYSECT представляет собой название программной секции или фиктивной области, содержащей макрокоманду.

Значением его является имя последнего из операторов START, CSECT или DSECT, который встречается перед макрокомандой. Если перед макрокомандой не появляются именованные операторы START, CSECT и DSECT, то параметр &SYSECT получает пустое знаковое значение.

Если в макроопределении используются операторы CSECT и DSECT, то они устанавливают значения параметру &SYSECT для любых последующих внутренних макрокоманд, но не изменяют значение &SYSECT в макроопределении, соответствующем внешней макрокоманде. Таким образом, значение &SYSECT для каждого обрабатываемого макроопределения является постоянным.

Параметр &SYSECT нужно использовать в макроопределении в том случае, если в операторах макроопределения присутствуют операторы CSECT и DSECT. По названным операторам из макроопределения создается программная секция или фиктивная область, поэтому макроопределение должно восстановить ту программную секцию, в которой находится макрокоманда. В этом случае операторы, находящиеся за макрокомандой, будут принадлежать к той секции, к которой относится и сама макрокоманда. Если же не восстанавливать программную секцию, к которой принадлежит макрокоманда, то команды, находящиеся за макрокомандой, будут принадлежать к той секции или фиктивной области, которая создается макроопределением. В этом случае нарушится программное деление на секции и программа окажется неправильной. Восстановить в макроопределении ту программную секцию, к которой принадлежит макрокоманда, можно с помощью оператора:

Название	Операция	Операнды
&SYSECT	CSECT	

В нижеприведенном примере в макроопределении генерируется фиктивная область, поэтому в макроопределении присутствует оператор, который, используя системный параметр &SYSECT, восстанавливает программную секцию.

Название	Операция	Операнды	Идентификация
A&SYSNDX &SYSECT	MACRO	макроопределение	1
	MC	&P	2
	USING	A&SYSNDX,&P	3
	L	1,A&SYSNDX	4
	DSECT		5
	DC	F'1'	6
	CSECT		7
	MEND		8

Название	Операция	Операнды	Идентификация
ST1	START		9
	BALR	11,0	10
	USING	*,11	11
	LA	12,F1	12
	MC	12 макрокоманда	13
	AR	1,1	14
	SVC	14	15
F1	DC	F'1'	16
	END	ST1	17

В результате генерации получится следующая программа:

Название	Операция	Операнды	Идентификация
ST1	START		9
	BALR	11,0	10
	USING	*,11	11
	LA	12,F1	12
	MC	12 макрокоманда	13
	USING	A0001,12	
	L	1,A0001	
A0001	DSECT		
ST1	DC	F'1'	
	CSECT		
F1	AR	1,1	14
	SVC	14	15
	DC	F'1'	16
	END	ST1	17

По оператору 5 макроопределения генерируется фиктивная область с именем A0001. Оператор 7 макроопределения восстанавливает программную секцию ST1, в которой находится макрокоманда MC, поэтому операторы 14, 15 и 16 принадлежат к программной секции ST1. Если бы в макроопределении не было оператора 7, то программная секция ST1 не восстанавливалась бы и операторы 14, 15 и 16 принадлежали к фиктивной области A0001.

4.4.3. Параметр &SYSLIST

Системный параметр &SYSLIST позволяет обращаться к операндам макрокоманды без использования названий постоянных параметров. Для обращения к операндам макрокоманды параметр &SYSLIST нужно записывать с индексами. С помощью записи &SYSLIST(*n*) можно обращаться к операнду макрокоманды с номером *n*. Индекс *n* может быть десятичным числом или арифметическим выражением, но его значение должно быть положительным. Индекс *n* не должен превышать числа операндов в макрокоманде. Если *n*=0, то это означает обращение к опущенному

операнду. Запись &SYSLIST (n, m) используется для обращения к элементу номер m списка, который является операндом номер n макрокоманды.

Ниже приведена макрокоманда, к операндам которой можно обращаться с помощью параметра &SYSLIST.

Название	Операция	Операнды
	МCOM	AREA,,(M1,M2,,M4,M5),P

Для этой макрокоманды значение параметра &SYSLIST(4) означает обращение к операнду P, &SYSLIST (3,2) — обращение к элементу M2 списка, который является третьим операндом макрокоманды, &SYSLIST(2) — обращение ко второму операнду, который в макрокоманде опущен.

Таким образом, операнд макрокоманды можно использовать или с помощью названия постоянного параметра, или с помощью параметра &SYSLIST. Последний выгодно применять в случае, когда названия постоянных параметров не указаны в операторе прототипа (например, если неизвестно, сколько операндов будет записано в макрокоманде). Невозможно обойтись без параметра &SYSLIST и в тех случаях, когда некоторый оператор макроопределения в зависимости от значений операнда макрокоманды должен использовать тот или иной операнд макрокоманды. Например, параметр макрокоманды указывает номер операнда макрокоманды, который необходим при обработке макроопределения: в одном случае это будет первый операнд, в другом — третий операнд. В операторе макроопределения используется запись &SYSLIST(n), и в первом случае n присваивается значение 1, а во втором — 3.

4.5. ПЕРЕМЕННЫЕ ПАРАМЕТРЫ

В макроопределениях до сих пор рассматривались постоянные параметры, значения которых задаются макрокомандой, и системные параметры, значения которых устанавливаются транслятором к моменту обработки макрокоманды. Значения постоянных и системных параметров не изменяются на протяжении обработки всего макроопределения. Но в некоторых случаях необходимо иметь такие параметры, значения которых при генерации изменяются. Для этой цели используются переменные параметры, значения которых можно изменять специальными командами генерации. Как и постоянные параметры, переменные параметры при обработке макроопределений заменяются соответствующими им значениями. В некоторых случаях необходимо, чтобы значением переменного параметра было целое число, в других случаях — набор знаков, в третьих — логическое значение «истина» или «ложь». В связи с

этим используются три типа переменных параметров: арифметические, знаковые и логические.

Переменные параметры можно использовать во всех операторах как в макроопределениях, так и вне их (в основной части программы). В зависимости от места действия переменные параметры подразделяются на глобальные и локальные. Глобальные переменные параметры (они рассматриваются в разд. 4.7.1) передают значения для операторов всей программы: во всех макроопределениях и вне их. Локальные переменные параметры передают значения операторам в одном и том же макроопределении или операторам вне макроопределений.

Все локальные переменные параметры, которые нужно использовать в данном макроопределении (в основной части программы), необходимо перечислить в командах определения локальных переменных параметров, которые записываются в этом же макроопределении (в основной части программы).

Арифметические параметры определяются оператором LCLA, знаковые параметры — оператором LCLC, логические параметры — оператором LCLB. Формат этих операторов приведен ниже.

Название	Операция	Операнды
Пробел	LCLA или LCLC или LCLB	Имена переменных параметров, разделенные запятыми

В поле операндов записываются имена тех локальных переменных параметров, которые нужно использовать в данном макроопределении или в основной части программы. Операторы LCLA, LCLC и LCLB в макроопределении должны записываться сразу же за оператором прототипа, а в основной части программы они должны быть самыми первыми за всеми программными макроопределениями (между макроопределениями и этими операторами могут записываться операторы PRINT, TITLE, EJECT, SPACE и комментарии).

Правила записи переменных параметров совпадают с правилами записи постоянных параметров. Переменные параметры можно использовать только в том макроопределении (или в основной части программы), в котором они определены. Для других макроопределений эти параметры как бы не существуют.

Операторы LCLA, LCLC, LCLB не только определяют переменные параметры, но и задают им начальные значения. Начальным значением арифметического параметра является число 0, знакового параметра — пустое знаковое значение (т. е. отсутствие знаков), логического параметра — логическое значение 0 («ложь»).

В следующем примере рассматривается определение переменных параметров.

Название	Операция	Операнды	Идентификация
&PN	MACRO		1
	M45	&P1,&P2	2
	LCLB	&PB1	3
	LCLA	&PA1,&P1,&PB1,&PB2	4
	L	&PA1,&P1	5
	LCLC	&C1	6
	MEND		7

В этом макроопределении можно использовать логический параметр &PB1 и арифметические параметры &PA1 и &PB2. Арифметические параметры &P1 и &PB1 оператором 4 определяются неправильно, потому что параметр &P1 объявлен в прототипе постоянным параметром, а параметр &PB1 уже определен как логический. Знаковый параметр &C1 в операторе 6 также определяется неверно, так как перед этим оператором LCLC находится оператор 5, являющийся машинной командой. Если переставить местами операторы 5 и 6, то параметр &C1 будет определяться правильно.

4.5.1. Арифметические параметры

Арифметические параметры позволяют производить действия над целыми числами. Значение арифметического параметра можно изменять оператором SETA. Его формат приведен ниже.

Название	Операция	Операнды
Арифметический параметр	SETA	Арифметическое выражение

Значение арифметического выражения, записанного в поле операндов, присваивается арифметическому параметру, записанному в поле названия. Арифметическое выражение представляет собой один терм или комбинацию термов, связанных арифметическими операциями сложения (+), вычитания (-), умножения (*) и деления (/).

Для указания порядка, в котором должны выполняться арифметические действия, могут использоваться скобки. Выражение не должно начинаться со знака операции. Значением выражения является целое число со знаком, которое может находиться в пределах от -2^{31} до $2^{31}-1$.

В арифметическом выражении можно использовать: самоопределенные термы, параметры, характеристики.

На использование самоопределенных термов не накладываются никаких ограничений. Например, правильными являются выражения, записанные во всех следующих операторах SETA:

Название	Операция	Операнды
&A1	SETA	5
&A2	SETA	$X'A' - B'11\emptyset\emptyset'$
&A3	SETA	$2*(C' \&\&' - X'\emptyset A' * (C'1' + B'11'))$

В этом примере параметру &A1 присваивается значение +5, параметру &A2 — значение —2, параметру &A3 — значение —4740. Если арифметический параметр употребляется в арифметических выражениях, то используется его арифметическое значение (например, в поле операндов оператора SETA). В остальных случаях значение арифметического параметра преобразуется в знаковое значение, которое получается преобразованием значения параметра в целое число без знака с отбрасыванием первых нулей.

Постоянные параметры в арифметических выражениях можно использовать только в том случае, если их значениями являются самоопределенные термы (т. е. в макрокоманде параметру соответствует самоопределенный терм). При вычислении выражений параметры заменяются соответствующими им значениями. В следующем макроопределении в арифметических выражениях используются параметры. В макрокоманде показаны значения, которые соответствуют постоянным параметрам.

Название	Операция	Операнды	Идентификация
	MACRO	макроопределение	1
	MA1	&P1, &P2	2
	LCLA	&A11, &NA1	3
&A11	SETA	&P1 + 2	4
&NA1	SETA	&A11 * 2 — 3 \emptyset	5
&A11	SETA	&A11 + 1 \emptyset	6
&NA1	SETA	&P2	7
	MEND		8
	MA1	$X'A'5 - A$ макрокоманда	9

Оператор 4 присваивает параметру &A11 значение +12, потому что параметру &P1 соответствует значение $X'A'$. Оператор 5 присваивает параметру &NA1 значение —6, так как значение параметра &A11 в этот момент равно 12. В поле операндов оператора 6 используется значение параметра &A11, равное 12. Оператор 6 увеличивает это значение на 10, и параметру &A11 присваивается новое значение 22. В операторе 7 в качестве терма выражения нельзя использовать параметр &P2, потому что его значением в макрокоманде является не самоопределенный терм, а выражение 5—A.

В качестве термов арифметического выражения можно употреблять некоторые характеристики постоянных параметров или простых символических имен. Характеристика — это целое число или буква, отражающая свойства простого символического име-

ни или операнда макрокоманды, который соответствует постоянному параметру. В выражениях макроопределения можно использовать характеристики только постоянных параметров, а в основной части программы — только простых символических имен. Если в макроопределении приводится характеристика постоянного параметра, то это значит, что используется характеристика того операнда макрокоманды, который соответствует постоянному параметру.

Рассмотрим характеристики длины, количества знаков и количества операндов, которые можно применять в арифметических выражениях. Характеристика длины простого имени описывалась в разд. 2.1.4. Значением характеристики длины является количество байт области памяти, которая определяется этим именем. Характеристика длины записывается как буква L с апострофом, за ними следует параметр или простое имя, характеристика которого используется, например L'&NAME, L'AB1. Можно применять характеристику длины только тех постоянных параметров, которым в макрокоманде соответствуют простые символические имена. В следующем примере приведены простые символические имена, которые определяют различные области памяти.

Название	Операция	Операнды
N1	MVC	A,N1
D2	DS	ØDL5'2'
D4	DC	CL(5—2)'AB'

Характеристика длины этих операторов имеет такие значения:
L'N1=6 L'D2=5

Характеристику длины имени D4 в макросредствах употреблять нельзя, потому что длина этой константы задана выражением. Характеристику длины имени константы в арифметических выражениях можно использовать только тогда, когда длина константы задана десятичным термом и в поле операндов DC или DS нет параметров. В следующем примере используется характеристика длины постоянных параметров.

Название	Операция	Операнды	Идентификация
&A1 &A2 &A3 &A2	MACRO	макроопределение	1
	M1	&P1,&P2,&P3	2
	LCLA	&A1,&A2,&A3	3
	SETA	L'&P1+1	4
	SETA	L'&P2	5
	SETA	L'&P3	6
	SETA	L'&A1	7
K D	MEND		8
	M1	K,D,5 макрокоманда	9
	SVC	14	10
	DC	FL(3—2)'1'	11
	END		12

Параметру &P1 в макрокоманде соответствует имя K, которое именует команду SVC длиной 2 байта. Поэтому параметру &A1 оператором 4 присваивается значение 3. Параметру &P2 в макрокоманде соответствует простое имя D, но использовать характеристику длины этого параметра при том значении, которое соответствует ему в макрокоманде, нельзя. Длина константы с именем D, соответствующим параметру &P2, задана выражением. Нельзя также использовать для данной макрокоманды характеристику длины параметра &P3, потому что ему в макрокоманде соответствует самоопределенный терм. Оператор 7 в макроопределении записан неправильно: можно употреблять характеристики только постоянных параметров.

Характеристика количества знаков применяется только в макроопределении. Значением ее является количество знаков в операнде, который соответствует постоянному параметру. Если последнему соответствует опущенный операнд, то характеристика количества знаков равна нулю. Характеристика количества знаков обозначается буквой K с апострофом, за ним записывается тот постоянный параметр, характеристика которого используется. В примере рассматривается макроопределение и соответствующая ему макрокоманда.

Название	Операция	Операнды	Идентификация
&A1 &A2	MACRO	макроопределение	1
	XK1	&P1,&P2	2
	LCLA	&A1,&A2	3
	SETA	K'&P1	4
	SETA	K'&P2+1	5
	DC	F'&A1,&A2'	6
	MEND		7
	XK1	A1@1,X'AB' макрокоманда	8
	XK1	(1,2,4) макрокоманда	9

При обработке макроопределения для оператора 8 оператор 4 присваивает арифметическому параметру &A1 значение 4, потому что количество знаков в операнде, соответствующем параметру &P1, равно 4. Параметру &A2 оператором 5 присваивается значение 6, так как K'&P2=5. Для оператора 8 генерируется следующий оператор:

Название	Операция	Операнды
	DC	F'4,6'

Когда переменные параметры в операторе 6 заменяются своими значениями, то знак чисел отбрасывается. В операторе 9 значением параметра &P1 является список, состоящий из семи зна-

ков, поэтому арифметическому параметру &A1 присваивается значение 7. Значение параметра &P2 — опущенный операнд, поэтому $K' \&P2 = 0$. Параметру &A2 присвоится значение 1. Для такой макрокоманды генерируется оператор:

Название	Операция	Операнды
	DC	F'7,1'

В макроопределении арифметическим параметрам &A1 и &A2 присваиваются арифметические значения, а потом эти значения используются в операторе 6 как значения параметров &A1 и &A2. Неправильно было бы выбросить из макроопределения операторы 4 и 5, а в операторе 6 вместо параметров &A1 и &A2 сразу написать соответствующие им выражения:

Название	Операция	Операнды
	DC	F'K'&P1, K'&P2+1'

Это неправильно потому, что арифметические выражения транслятор высчитывает только в тех полях операторов, в которых разрешена их запись, например в поле операндов оператора SETA. В остальных случаях параметры только заменяются на значения, им соответствующие. В результате при обработке приведенного оператора для первой макрокоманды был бы построен следующий оператор DC:

Название	Операция	Операнды
	DC	F'K'A121, K'X'AB'+1'

Характеристику количества операндов можно использовать только в макроопределении. Значением характеристики количества операндов является количество элементов в списке, соответствующем постоянному параметру. Если операнд не является списком, то значение этой характеристики принимается равной 1, а если операнд опущен, то значением характеристики считается 0. Эта характеристика записывается в виде буквы N с апострофом, за ним следует постоянный параметр, характеристика которого используется. Рассмотрим следующий пример.

Название	Операция	Операнды	Идентификация
&AP1 &AP2	MACRO	макроопределение	1
	XN1	&P1,&P2,&P3	2
	LCLA	&AP1,&AP2	3
	SETA	&P1+N'&P2-1	4
	SETA	N'&P3	5
	LM	&P1,&AP1,=A&P2	6
	MEND		7
* макрокоманды			8
	XN1	2,(1,2,15,21),A	9
	XN2	11,(1,1344)	10

Параметру &P2 в операторе 9 соответствует список, состоящий из четырех элементов, поэтому $N' \&P2 = 4$. Оператором 4 параметру &AP1 присваивается значение 5. Оператором 5 параметру &AP2 устанавливается значение 1. Для оператора 9 генерируется макрорасширение:

Название	Операция	Операнды
	LM	2,5,=A(1,2,15,21)

При обработке макроопределения для оператора 10 параметру &AP1 присвоится значение 12, а параметру &AP2 — значение 0, поэтому сгенерируется такое макрорасширение:

Название	Операция	Операнды
	LM	11,12,=A(1,1344)

В макроопределении можно вместо названия постоянного параметра использовать системный параметр &SYSLIST. Характеристики этого параметра могут быть такие же, как характеристики постоянных параметров. Например, запись $N' \&SYSLIST(2)$ обозначает характеристику количества операндов второго постоянного параметра. Для последнего рассматриваемого макроопределения $N' \&SYSLIST(2) = N' \&P2$. Но запись $N' \&SYSLIST$ имеет самостоятельное значение. Она обозначает количество операндов в макрокоманде. В нижеприведенном макроопределении используется такая характеристика:

Название	Операция	Операнды
&A1	MACRO	макроопределение
	MS1	&R1,&P1
	LCLA	&A1
	SETA	N'&SYSLIST
	LM	&R1,&R1+&A1,&P1
	MEND	
	MS1	1,A1,A2 макрокоманды
	MS1	5,A1,A2,A3,A4,,A6,A7

В первой макрокоманде записаны три операнда, поэтому параметру &A1 оператором SETA присваивается значение 3. В результате для этой макрокоманды сгенерируется макрорасширение:

Название	Операция	Операнды
	LM	1,1+3,A1

Во второй макрокоманде записано восемь операндов (один из них опущенный), поэтому параметру &A1 присвоится значение 8 и сгенерируется макрорасширение другого вида:

Название	Операция	Операнды
	LM	5,5+8,A1

Арифметические параметры можно использовать не только в макроопределениях, но и в основной части программы, что демонстрируется на следующем примере.

Название	Операция	Операнды	Идентификация
&A A&SYSNDX	MACRO	макроопределение	1
	M1	&P1	2
	LCLA	&A	3
	SETA	K'&P1	4
	DC	F'&A'	5
	MEND		6
*		основная часть программы	7
&A	LCLA	&A	8
	SETA	Ø—L'DK1	9
DK1	M1	&A макрокоманда	10
	DC	D'123'	11
	END		12

Параметры &A, записанные в макроопределении и в основной части программы, — разные параметры. Каждый из них определяется отдельным оператором LCLA в макроопределении и в основной части программы, а их значения изменяются соответствующими операторами SETA. Оператор 9 в основной части программы присвоит параметру &A значение —8, потому что длина константы DK1 равна восьми байтам. Значением операнда макрокоманды будет число 8 (знак «—» отбрасывается), поэтому оператором 4 параметру &A присвоится значение 1, которое используется в операторе 5. В результате трансляции в программе окажутся следующие константы:

Название	Операция	Операнды	Идентификация
AØØØ1	DC	F'1'	11
DK1	DC	D'123'	

4.5.2. Знаковые параметры

С помощью знаковых параметров можно производить действия над любыми знаками кода ДКОИ. Первоначальное значение знакового параметра, которое устанавливается оператором LCLC, можно изменить оператором SETC. Его формат:

Название	Операция	Операнды
Знаковый параметр	SETC	Знаковое выражение

Значение знакового выражения, записанного в поле операндов оператора SETC, присваивается знаковому параметру, записанному в поле названия. Знаковым выражением может быть один терм или несколько термов, связанных операцией соединения (.). Значением выражения является набор знаков. Знаковому параметру по оператору SETC присваивается значение, содержащее не более восьми знаков. Если длина знакового выражения в поле операндов больше восьми знаков, то лишние знаки отбрасываются. В знаковом выражении можно использовать следующие термы: знаковую строку, подстроку знаков, характеристику типа.

Знаковая строка — это набор любых знаков, заключенный в апострофы, например:

'A112'
'C+D.(1,2)'
'A+B'

Значением знаковой строки являются все знаки, исключая апострофы, в которые заключены эти знаки. Если в знаковую строку нужно включить апостроф или знак &, то их нужно запи-

сывать как два апострофа или два знака & соответственно. При этом в значение знакового выражения попадет один апостроф, но два знака &, например:

Название	Операция	Операнды
&C1 &C2	SETC SETC	'C'A' 'A+C'&&''

Параметру &C1 присваивается значение C'A', параметру &C2 — значение A+C'&&'.

В знаковой строке может присутствовать параметр, тогда при вычислении значения выражения параметр заменяется всеми знаками, которые ему соответствуют. Знак арифметического параметра при этом опускается. Если в значение параметра, который присутствует в знаковой строке, входят апострофы, то в значение знаковой строки попадает каждый апостроф, например:

Название	Операция	Операнды
&A1 &C1	MACRO	макроопределение
	MZI	&P1
	LCLA	&A1
	LCLC	&C1
	SETA	Ø—6
	SETC	'AB&A1&P1'
	MEND	
	MZI	+X'5' макрокоманда
	END	

Арифметическому параметру &A1 присваивается значение —6. Постоянному параметру &P1 соответствует значение +X'5', поэтому знаковому параметру &C1 присваивается значение AB6+X'5'.

В знаковом выражении можно использовать подстроку знаков, (подстрока знаков — это часть знаковой строки). Для этого записывают знаковую строку, из которой выделяется подстрока, и указывают с помощью двух арифметических выражений, какие знаки входят в подстроку. Арифметические выражения записываются в скобках и отделяются друг от друга запятой. Первое выражение указывает номер знака, с которого начинается подстрока, а второе — количество знаков, входящих в подстроку (апострофы, в которые заключена знаковая строка, при подсчете значения подстроки не учитываются). Например, значение подстроки 'A12B34C567DEF'(4,7) равно B34C567.

Если второе выражение указывает количество знаков большее, чем их есть в знаковой строке, то значение подстроки составят только имеющиеся знаки. Например, значение подстроки 'ABC123F689KLM'(8,7) равно 689KLM.

В следующем макроопределении в качестве терма знакового выражения используется подстрока знаков.

Название	Операция	Операнды
&C1	MACRO	макроопределение &P1,&P2 &C1 'K&P1'(&P2,&P2+4) 5,&C1 A+D—C'&&', 4 макрокоманда
	ZP1	
	LCLC	
	SETC	
	L	
	MEND	
	ZP1	

Знаковой строкой, из которой выделяется подстрока знаков, является строка 'KA+D—C'&&', потому что параметру &P1 соответствует значение A+D—C'&&'. Из этого набора знаков выделяется подстрока знаков. Значением первого арифметического выражения, записанного в операторе SETC для выделения подстроки, является число 4, значением второго выражения — число 8. Поэтому значением подстроки будут знаки D—C'&&', которые являются значением параметра &C1. В макрорасширение генерируется такая команда:

Название	Операция	Операнды
	L	5,D—C'&&'

В знаковом выражении в качестве терма можно употреблять характеристику типа. Она записывается буквой Т с апострофом, за которым следует параметр или простое имя, чья характеристика используется. Значением характеристики всегда является буква.

В макроопределениях можно применять характеристику типа только постоянного параметра, в основной части программы — только простого символического имени. Это имя может называться любой оператор языка Ассемблера или определяться оператором EXTRN как внешнее имя.

Если операнд макрокоманды — самоопределенный терм, то значением характеристики типа является буква N. Для опущенного операнда значением характеристики типа принята буква O. В примере рассматривается характеристика типа таких операндов.

Название	Операция	Операнды
	MACRO	макроопределение &P1,&P2 &C1,&C2
	MT1	
	LCLC	

Название	Операция	Операнды
&C1 &C2	SETC SETC DC MEND MT1 MT1	T'&P1 T'&P2 C'&C1&C2' ,C'K' макроккоманды 5,X'BC'

В первой макрокоманде постоянному параметру &P1 соответствует опущенный операнд, поэтому знаковому параметру &C1 присваивается буква O. Значением параметра &P2 является самоопределенный терм C'K', поэтому параметру &C2 присваивается значение N. В результате сгенерируется макрорасширение вида:

Название	Операция	Операнды
	DC	C'ON'

Во второй макрокоманде операндами являются самоопределенные термы, для нее сгенерируется макрорасширение:

Название	Операция	Операнды
	DC	C'NN'

Значением постоянного параметра может быть простое символическое имя. Это простое имя может называть оператор DC, машинную команду, макрокоманду и другие операторы. Характеристике типа присваивается значение в зависимости от того, какой оператор называет простое имя. Характеристики типа различаются также в тех случаях, когда символические имена именуют операторы DC, определяющие разные типы констант, а также когда оператор DC содержит параметры или когда длина константы задана выражением. Например, имена чисел с фиксированной и плавающей точкой имеют разные характеристики. В табл. 15 приведены значения характеристики типа для имен, называющих различные операторы.

Характеристика типа простых имен, определяемых оператором EXTRN, имеет значение T, характеристика типа всех остальных видов операндов макрокоманды — значение U. Например, значение U имеет характеристика типа литерала, имени операторов EQU и LTORG, имени, называющего оператор DC, в котором содержатся параметры, неопределенного имени. Если имя называет оператор

DC, модификатор длины которого задан не десятичным термом, а выражением, то характеристика типа такого имени тоже имеет значение U.

Таблица 15

Значение характеристики типа	Оператор или тип константы
I	Машинная команда
J	Операторы CSECT, DSECT, START
M	Макрокоманда
W	Оператор CCW
A	Адресная константа типа A с неявно заданной длиной
B	Двоичная константа
C	Знаковая константа
D	Константа типа D, неявная длина
E	Константа типа E, неявная длина
F	Константа типа F, неявная длина
H	Константа типа H, неявная длина
G	Константа с фиксированной точкой, явная длина
K	Константа с плавающей точкой, явная длина
P	Упакованная десятичная константа
R	Адресная константа, явная длина
S	Адресная константа типа S, неявная длина
V	Адресная константа типа V, неявная длина
X	Шестнадцатеричная константа
Y	Адресная константа типа Y, неявная длина
Z	Распакованная десятичная константа

В следующем примере используется характеристика типа простых символических имен и параметров.

Название	Операция	Операнды	Идентификация
	MACRO	макроопределение	1
	MTP	&P1,&P2,&P3	2
	LCLC	&C1,&C2,&C3	3
&C1	SETC	T'&P1	4
&C2	SETC	T'&P2	5
&C3	SETC	T'&P3	6
	DC	C'&C1&C2&C3'	7
	MEND		8
	LCLC	&C1,&C2,&C3,&C4,&C5, &C6,&C7,&C8,&C	9
ST1	START	, основная часть программы	10
&C1	SETC	T'ST1	11
&C2	SETC	T'NP	12
&C3	SETC	T'NEX	13
&C4	SETC	T'NEQU	14
&C5	SETC	T'DCØ	15
&C6	SETC	T'DC1	16
&C7	SETC	T'DC2	17

Название	Операция	Операнды	Идентификация
&C8	SETC	T'DC3	18
	DC	C'&C1&C2&C3&C4'	19
	DC	C'&C5&C6.&C7.&C8'	20
	MTP	X'AB',=F'11' макрокоманда	21
DCØ	DC	F'1'	22
DC1	DC	FL4'1'	23
DC2	DC	FL(6—1)'1'	24
DC3	DC	F&C'1'	25
	EXTRN	NEX	26
NEQU	EQU	ST1	27
	END		28

Параметру &P1 в макрокоманде соответствует самоопределенный терм X'AB', поэтому оператором 4 параметру &C1 в макроопределении присваивается значение N. Оператором 5 параметру &C2 в макроопределении присваивается значение U, так как параметру &P2 в макрокоманде соответствует литерал. Значением параметра &P3 является опущенный операнд, поэтому знаковому параметру &C3 присваивается значение O. В результате по макрокоманде из макроопределения сгенерируется оператор:

Название	Операция	Операнды
	DC	C'NUO'

В основной части программы также используются знаковые параметры &C1, &C2, &C3, но им присваиваются другие значения соответственно J, U, T. Параметру &C4 дается значение U, потому что имя NEQU именует оператор EQU. Знаковым параметрам &C5, &C6, &C7, &C8 в основной части программы присваиваются значения F, G, U, U. Из операторов 19 и 20 в основной части программы сгенерируются такие операторы:

Название	Операция	Операнды
	DC	C'JUTU
	DC	C'FGUU

Таким образом, после генерации получится следующая программа:

Название	Операция	Операнды	Идентификация
ST1	START		1Ø
	DC	C'JUTU'	19
	DC	C'FGUU'	2Ø
	MTP	X'AB',=F'1' макрокоманда	21
	DC	C'NUO'	
DCØ	DC	F'1'	22
DC1	DC	FL4'1'	23
DC2	DC	FL(6—1)'1'	24
DC3	DC	F'1'	25
	EXTRN	NEX	26
NEQU	EQU	ST1	27
	END		28

Знаковое выражение можно задавать в виде комбинации термов, которые связаны операцией соединения (.). Но если в выражении присутствует характеристика типа, то других термов в этом выражении быть не должно. Выполнение операции соединения заключается в простом присоединении знаков второго терма к знакам первого терма. В следующем макроопределении в качестве знакового выражения дана комбинация термов:

Название	Операция	Операнды
	MACRO	макроопределение
	VZ1	&P1,&P2,&P3
	LCLC	&C1,&C2
	LCLA	&A1
&A1	SETA	&P1*2+3
&C1	SETC	T'&P2
&C2	SETC	'&P3'(&P1,&A1).&C1'
	DC	C'&C2'
	MEND	
	VZ1	2,7+BC,A13" макрокоманда

Арифметическому параметру &A1 присваивается значение 7, а знаковому параметру &C1 — значение U, потому что значением параметра &P2 в макрокоманде является набор знаков 7+BC. Знаковому параметру &C2 присваивается значение 13"U, так как значение параметра &P3 — набор знаков A13". Значением знаковой строки '&P3' являются знаки A13". Из этой строки, начиная со второго знака, нужно выбрать 7 знаков, а так как их всего в строке 5, то берутся только имеющиеся, поэтому значением подстроки '&P3'(&P1,&A1) являются знаки 13". В результате сгенерируется такое макрорасширение:

Название	Операция	Операнды
	DC	C'13"U'

В данном примере можно обойтись без арифметического параметра &A1, но тогда выражение в операторе SETC для присваивания значения параметру &C2 необходимо записать в следующем виде: '&P3'(&P1, &P1*2+3). '&C1'. Значение знакового выражения от этого не изменится.

За подстрокой знак соединения записывать не обязательно. Например, выражение '&P1'(2,2). 'AB' можно записать так: '&P1'(2,2)'AB'.

4.5.3. Логические параметры

С помощью логических параметров обрабатываются логические значения. Первоначальное значение логического параметра, которое устанавливается оператором LCLB, можно изменять оператором SETB. Его формат:

Название	Операция	Операнды
Логический параметр	SETB	Логическое выражение, заключенное в скобки

Значение логического выражения, записанного в поле операндов оператора SETB, присваивается логическому параметру, записанному в поле названия. Логическим выражением может быть один терм или несколько термов, связанных логическими операциями. Для указания порядка, в котором должны выполняться действия, используются скобки. В качестве логических операций применяются операции AND, OR, NOT (т. е. И, ИЛИ, НЕТ). Значением логического выражения является логическое значение 1 («истина») или 0 («ложь»). В логическом выражении можно использовать следующие термы: логическое значение 1, логическое значение 0, логический параметр, арифметическое отношение, знаковое отношение. Если в качестве термов участвуют логические значения 1 или 0, то они заключаются в скобки.

Термом логического выражения может быть логический параметр. Значением логического параметра является логическое значение 0 или 1, это значение и участвует при вычислении выражения. В следующем примере приведены логические выражения:

Название	Операция	Операнды
	MACRO	
	ML1	&P1
	LCLB	&B1,&B2,&B3,&B4
&B1	SETB	(Ø)
&B2	SETB	(1)
&B3	SETB	(&B2)
&B4	SETB	(&B1)
	MEND	

Значением параметра &B1 и &B4 является логическое значение 0, значением &B2 и &B3 — логическое значение 1. В логическом выражении можно использовать арифметические и знаковые отношения.

Арифметическое отношение — это два арифметических выражения, которые связаны операцией отношения. Операциями отношения могут быть: EQ (равно), NE (не равно), LT (меньше), GT (больше), LE (меньше или равно), GE (больше или равно).

При записи перед операцией отношения и после нее нужно пропустить хотя бы один пробел. Но если логической операции предшествует (или следует за ней) специальный знак (скобка, апостроф или знак &), то операцию отношения можно записывать сразу за специальным знаком (сразу перед ним), не оставляя пробела. При вычислении отношения вычисляются и сравниваются составляющие его арифметические выражения. После этого проверяется, подходит ли операция отношения для вычисленных значений выражений. Если значения выражений удовлетворяют операции отношения, то арифметическому отношению соответствует логическое значение 1, иначе — 0.

В следующем примере используются арифметические отношения:

Название	Операция	Операнды
	MACRO	
	ML2	макроопределение
	LCLB	&P1,&P2
&B1	SETB	&B1,&B2
&B2	SETB	(&P1 EQ 7)
	MEND	((&P2+1)*2 LT 11)
	ML2	6,X'2' макрокоманда

Параметру &P1 соответствует в макрокоманде значение 6. Это значение не равно 7, поэтому логическому параметру &B1 присваивается логическое значение 0. Значением параметра &P2 является X'2', следовательно, значение выражения (&P2+1)*2 равно 6 (меньше 11), поэтому значением параметра &B2 является логическое значение 1.

Знаковым отношением называются два знаковых выражения, связанные операцией отношения. При составлении знаковых отношений используются те же операции отношения, что и при составлении арифметических отношений. В знаковых отношениях сравниваются наборы знаков. При определении значения знакового отношения сначала вычисляются значения знаковых выражений, которые составляют знаковое отношение, а затем сравниваются двоичные коды знаков. Знаки для сравнения представляются в коде ДКОИ. Если в сравнении участвуют строки знаков разной длины, то более длинная строка считается большей. Значение знакового отношения определяется по результату операции так же, как и значение арифметического отношения. Рассмотрим пример использования знаковых отношений:

Название	Операция	Операнды
	MACRO	макроопределение
	ML3	&P1,&P2
	LCLB	&B1,&B2,&B3
&B1	SETB	(T'&P1 EQ 'F')
&B2	SETB	('&P2' (1,2) EQ 'X''')
&B3	SETB	('&P1' LE '&P2')
	MEND	
DCN	ML3	DCN,X'ABC' макрокоманда
	DC	D'I'
	END	

Характеристикой типа параметра &P1 является буква D, поэтому параметру &B1 присваивается логическое значение 0. Первые два знака значения параметра &P2 являются знаками X', в связи с чем параметру &B2 присваивается логическое значение 1. Значение параметра &B3 также 1, так как значение параметра &P1 меньше значения &P2.

Логические выражения составляются с помощью логических операций AND, OR, NOT. При записи логического выражения перед логическими операциями и за ними необходимо записывать хотя бы один пробел. Как и при записи отношений, пробелы записывать не обязательно, если непосредственно перед операцией или после нее присутствует специальный знак. Операция NOT выполняется над отдельным логическим значением, а операция AND и OR — над двумя логическими значениями. При выполнении операции NOT логическое значение изменяется на противоположное (0 на 1 и наоборот). Результаты выполнения операций AND и OR над термами, имеющими разные значения, приведены в табл. 16.

При вычислении логического выражения первой выполняется операция NOT, затем AND и последней OR. Если в логическом выражении даны скобки, то первым вычисляется выражение, заключенное в скобки. В логическом выражении можно записывать подряд две операции, если первой из них является операция

Таблица 16

Значение термина		Результат операции	
первого	второго	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

AND или OR, а второй — NOT. Применение этих правил иллюстрируется следующим примером.

Название	Операция	Операнды	Идентификация
&B2	LCLB	&B1,&B2,&B3	1
&B3	SETB	(1)	2
&B3	SETB	(&B1 OR &B2 AND NOT &B1)	3
&B3	SETB	((T'D1 EQ 'D' OR &B1 AND &B2))	4
D1	DC	A(*)	5
	END		6

Оператор LCLB устанавливает логическим параметрам &B1, &B2 и &B3 первоначальное значение 0. Оператор 3 присваивает параметру &B3 значение 1, а оператор 4 — значение 0.

4.6. ГЕНЕРАТОРНЫЕ ПЕРЕХОДЫ

Рассмотрим следующий пример:

Название	Операция	Операнды	Идентификация
	MACRO	макроопределение	1
	M1	&P1,&P2	2
	AR	1,&P1	3
	AR	1,&P2	4
	MEND		5
	M1	21,22 макрокоманда	6
	END		7

Для макрокоманды в данном случае сгенерируется макрорасширение вида:

Название	Операция	Операнды
	AR	1,21
	AR	1,22

Машинные команды здесь неправильны, потому что нет общих регистров с номерами 21 и 22. Ошибка возникла из-за того, что были неправильно определены операторы макрокоманды. Следовательно, в макроопределении нужно всегда проверять, правильно ли записаны операторы макрокоманды. Если операторы неправильны, то необходимо сразу же прекратить обработку макроопределения. Обработку можно прекратить, если после прототипа сразу обрабатывать оператор MEND, т. е. выполнить генераторный переход на оператор MEND. Проверить операторы макрокоманды и сделать переход на оператор MEND позволяют специальные операторы.

Генераторные переходы принципиально отличаются от переходов, которые осуществляются машинными командами, такими, как BAL, BALR, B. Машинные переходы выполняются при работе уже оттранслированной программы, при решении задачи. Действие машинного перехода заключается в передаче управления команде рабочей программы и выполнении этой командой своего действия. Например, если с помощью команды BZ сделан переход на команду AR, то начинает работать именно эта команда AR, т. е. выполняется сложение регистров.

Действие генераторного перехода заключается в том, что транслятор продолжает генерацию программы с того оператора, на который сделан генераторный переход. Например, если сделан генераторный переход на команду AR, то это значит, что команда AR генерируется транслятором в программу для дальнейшей обработки. Генераторные переходы, как и переходы, выполняемые машинными командами, разделяются на условные и безусловные.

Оператор, на который делается генераторный переход, должен иметь имя. Имя перехода записывается в поле названия оператора, на который нужно сделать переход. Имя перехода состоит из точки, за которой следует от одной до семи букв или цифр, первой за точкой должна быть буква.

Правильные имена перехода:

.A .A1BCD75 .L123B17

Неправильные имена перехода:

AB1	(первый знак не точка);
.2A	(после точки не буква);
.A1234567	(больше семи знаков после точки).

4.6.1. Оператор AIF

Оператор AIF проверяет некоторое условие и, в зависимости от его выполнения, осуществляет генераторный переход или на следующий оператор, или на оператор, названный именем перехода. Формат оператора AIF:

Название	Операция	Операнды
Имя перехода или пробел	AIF	Логическое выражение, заключенное в скобки, за которым следует имя перехода

Операторы AIF, записанные правильно:

Название	Операция	Операнды
.P3	AIF AIF AIF AIF	(&A1 LT 5).A1 (&B1 OR NOT &B2).P1 (&B1).P21 ('C' EQ 'X' AND T'C1 EQ'N').P31

Операторы AIF, записанные неправильно:

Название	Операция	Операнды	Идентификация
	AIF	(&B1 OR 5).A1	1
	AIF	(&B1)	2
	AIF	.P1	3
	AIF	(&B1 OR &B2).12	4

В операторе 1 неправильно записано логическое выражение (самоопределенный терм в логическом выражении может использоваться только в арифметическом отношении). В операторе 2 отсутствует имя перехода, а в операторе 3 не записано логическое выражение. В операторе 4 неправильно указано имя перехода.

При обработке оператора AIF вычисляется логическое выражение, записанное в поле операндов. Если значением выражения является логическое значение 1, то следующим обрабатывается оператор, названный именем перехода, т. е. происходит генераторный переход. Если выражение имеет значение 0, то следующим генерируется оператор, записанный после оператора AIF.

Оператор AIF позволяет генерировать нужную часть макроопределения. Например, макроопределение состоит из двух частей. Первая часть соответствует макрокоманде, в которой записаны два операнда, а вторая — макрокоманде, в которой записаны три операнда. В этом случае в макроопределении записывают оператор AIF, который проверяет, сколько операндов в макрокоманде. Если два операнда, то выполняется переход на те операторы макроопределения, которые обрабатывают два операнда макрокоманды, в противном случае генерируются операторы, которые обрабатывают три операнда.

Составим макроопределение, которое должно получать в нулевом регистре сумму двух чисел. Эти два числа находятся в регистрах, номера которых задаются параметрами. Значения операндов в макрокоманде, которые соответствуют параметрам, должны быть самоопределенными термами в пределах от 0 до 15. Оператор AIF в макроопределении должен проверять, правильно ли записаны операнды макрокоманды. Макроопределение, приведенное ниже, выполняет поставленную задачу.

Название	Операция	Операнды	Идентификация
	MACRO		1
	MSI	&P1,&P2	2
	AIF	(T'&P1 NE 'N') .M	3
	AIF	(T'&P2 NE 'N') .M	4
	AIF	(&P2 GT 15 OR &P1 GT 15) .M	5
	LR	Ø,&P1	6
	AR	Ø,&P2	7
.M	MEND		8

В логическом выражении оператора 3 проверяется, является ли операнд, соответствующий параметру &P1, самоопределенным термом. Если операнд — самоопределенный терм, то характеристикой типа параметра &P1 будет буква N. Следовательно, значение логического выражения оператора 3 равно 0, потому что операцией отношения записана операция NE (не равно). В этом случае следующим обрабатывается оператор 4. Если же параметру &P1 не соответствует самоопределенный терм, то значением выражения будет 1 и по оператору 3 произойдет переход на оператор, содержащий в поле названия имя перехода .M. Это имя содержит оператор MEND, поэтому следующим обрабатывается именно этот оператор, который указывает, что обработку макроопределения следует прекратить.

Оператор 4 проверяет, какую характеристику типа имеет операнд макрокоманды, соответствующий параметру &P2. Если значением &P2 не является самоопределенный терм, то происходит переход на оператор MEND, и обработка макроопределения прекращается. В противном случае обрабатывается оператор 5.

Логическое выражение оператора 5 состоит из двух арифметических отношений и использует логическую операцию OR. Значением первого отношения является значение 1, если $&P2 > 15$, и значение 0, если $&P2 \leq 15$. Значение второго отношения 1, если $&P1 > 15$, и 0, если $&P1 \leq 15$. Поэтому логическое выражение имеет значение 1, если $&P2 > 15$ или $&P1 > 15$, и значение 0, если $&P2 \leq 15$ и $&P1 \leq 15$. Таким образом, оператор 6 будет обрабатываться в том случае, если $&P2 \leq 15$ и $&P1 \leq 15$. Иначе выполняется генераторный переход на оператор MEND, и обработка макроопределения прекращается.

Таким образом, прежде чем начинается генерация машинных команд макроопределения, операторы AIF проверяют, правильно ли записаны операнды макрокоманды. Машинные команды строятся только в том случае, если операндами макрокоманды будут допустимые номера общих регистров. Предположим, записана макрокоманда:

Название	Операция	Операнды
	MS1	X'2,7

При таких операндах значением каждого логического выражения операторов 1, 2, 3 и 4 макроопределения является значение 0, поэтому сгенерируется следующее макрорасширение:

Название	Операция	Операнды
	LR AR	Ø, X'2' Ø, 7

Следует учесть, что операторы AIF не генерируются по макрокоманде в макрорасширение, т. е. размер программы, которая будет выполняться, не увеличится. Поэтому в программе можно многократно употреблять операторы AIF, не опасаясь ее увеличения. Использование операторов AIF помогает найти ошибки, допущенные в записи макрокоманд.

В рассматриваемом макроопределении операторы 3 и 4 проверяют, заданы ли операнды макрокоманды самоопределенными термами. Проверку может выполнить один оператор AIF, если логические выражения операторов 3 и 4 соединить в одно выражение с помощью операции OR. При этом в поле операндов оператора AIF записывается логическое выражение (T' & P1 NE 'N' OR T' & P2 NE 'N'). Значением его является 1, если хотя бы один терм имеет значение 1 (терм T' & P1 NE 'N' или терм T' & P2 NE 'N').

Однако в данном макроопределении нельзя объединить логические выражения операторов 3 и 5, т. е. нельзя записать выражение (T' & P1 NE 'N' OR & P2 GT 15 OR & P1 GT 15). Такое выражение невозможно использовать потому, что для некоторых значений параметров, заданных в макрокомандах, оно будет недействительным. Первое отношение этого выражения знаковое, а два других — арифметические. В последнем арифметическом отношении присутствует постоянный параметр & P1. Постоянный параметр в арифметических выражениях можно использовать только в том случае, если его значением является самоопределенный терм. Следовательно, если в макрокоманде в качестве значения & P1 будет записан не самоопределенный терм, то такое выражение будет неправильным и транслятор сообщит об этой ошиб-

ке. В таких случаях нужно записывать два оператора AIF. Первый из них проверяет, можно ли использовать параметры в арифметических выражениях. Если это условие выполняется, то второй оператор AIF с помощью арифметических отношений проверяет арифметические значения параметров.

Для того чтобы выполнить генераторный переход на некоторый оператор, необходимо в поле названия оператора записать имя перехода. Однако возможен случай, когда нужно сделать переход на оператор, в поле названия которого должен быть записан параметр или простое имя (например, оператор SETA). В этих случаях используется оператор ANOP. Его формат:

Название	Операция	Операнды
Имя перехода	ANOP	Не используется

Оператор ANOP при генерации не выполняет никакого действия, для него никакой оператор не генерируется. Поэтому если перед каким-либо оператором написать оператор ANOP и сделать генераторный переход на этот оператор; то оператор ANOP будет пропущен и дальше будет обрабатываться оператор, который записан за ANOP. Именно таким образом выполняется генераторный переход на те операторы, в поле названия которых невозможно записать имя перехода. Этот способ используется в следующем макроопределении.

Название	Операция	Операнды	Идентификация
.N1 &A1	MACRO		1
	MI		2
	LCLA	&A1	3
	ANOP		4
	SETA	&A1+1	5
	AR	Ø,&A1	6
	AIF	(&A1 LE 4).N1	7
	MEND		8

При обработке данного макроопределения пять раз будет сгенерирован оператор 6. Оператор 7 должен выполнять генераторный переход на оператор 5, который увеличивает значение параметра &A1. В поле названия оператора 5 записан параметр, поэтому переход выполняется на оператор 4 (ANOP), который записан перед оператором 5. Фактически при этом выполняется переход на оператор 5. По макрокоманде, соответствующей данному макроопределению, сгенерируются перечисленные ниже команды.

Название	Операция	Операнды
	AR	Ø,1
	AR	Ø,2
	AR	Ø,3
	AR	Ø,4
	AR	Ø,5

4.6.2. Оператор AGO

Формат оператора AGO:

Название	Операция	Операнды
Имя перехода или пробел	AGO	Имя перехода

Оператор AGO выполняет безусловный переход на оператор, названный именем перехода из поля операндов оператора AGO. Рассмотрим следующее макроопределение:

Название	Операция	Операнды	Идентификация
	MACRO		1
	MAI	&P1,&P2,&P3	2
	AIF	('&P3' EQ '—').MINUS	3
	L	Ø,=F'&P1'	4
	A	Ø,=F'&P2'	5
	AGO	.MEND	6
.MINUS	L	Ø,=F'&P1'	7
	S	Ø,=F'&P2'	8
.MEND	MEND		9

Это макроопределение осуществляет вычитание чисел, если третьим параметром является знак «—» и сложение, если третий параметр — любой другой знак.

Числа задаются операндами макрокоманды. Оператор 3 проверяет, какой знак соответствует параметру &P3. Если этим знаком является знак «—», то происходит генераторный переход на оператор 7 и генерируются команды, получающие разность чисел. Если проверяемый знак — любой другой знак, то генерируются команды, подсчитывающие сумму чисел. Оператор 6 используется для того, чтобы выполнить переход на оператор MEND для прекращения обработки макроопределения. Предположим, для рассматриваемого макроопределения записана такая макрокоманда:

Название	Операция	Операнды
	MAI	2,6,—

В этом случае сгенерируется макрорасширение вида:

Название	Операция	Операнды
	L S	$\emptyset, = F'2'$ $\emptyset, = F'6'$

Макроопределению может соответствовать и другая макрокоманда:

Название	Операция	Операнды
	MAI	2,6

Так как параметру &P3 соответствует опущенное значение, то для этой макрокоманды сгенерируется макрорасширение:

Название	Операция	Операнды
	L A	$\emptyset, = F'2'$ $\emptyset, = F'6'$

Операторы AIF и AGO, которые записаны в макроопределении, могут выполнять переход только на оператор, который находится в этом же макроопределении. Операторы AIF и AGO можно использовать вне макроопределений — в основной части программы. В этом случае они могут выполнять переход только на оператор, который находится в основной части программы.

4.6.3. Оператор ACTR

Во время генерации транслятор по операторам AIF и AGO выполняет генераторные переходы. Если неправильно записать эти операторы, может возникнуть генераторный цикл. Рассмотрим следующее макроопределение:

Название	Операция	Операнды	Идентификация
.A	MACRO		1
	ERI	&P1	2
	L	1,&P1	3
	AGO	.A	4
	MEND		5

При обработке данного макроопределения все время будет генерироваться оператор 3, потому что по оператору 4 на него всегда происходит генераторный переход. Для того чтобы транслятор мог предотвратить такие случаи, используется оператор ACTR. Его формат:

Название	Операция	Операнды
Пробел	ACTR	Арифметическое выражение

Арифметическое выражение, которое записывается в поле операндов, задает количество переходов, выполняемых по операторам AIF и AGO. При генерации транслятор не может выполнить генераторных переходов больше, чем разрешено оператором ACTR. Если делается попытка сделать лишний переход, то транслятор прекращает обработку макроопределения.

Оператор ACTR можно применять в макроопределениях и в основной части программы. При этом можно записывать только один оператор ACTR, который задает количество переходов соответственно в макроопределении или в основной части программы. Оператор ACTR должен находиться после операторов LCLA, LCLB, LCLC как в основной части программы, так и в макроопределении. Если оператор ACTR отсутствует, то принимается стандартное количество переходов, равное 150 для транслятора Ассемблера Е или 4096 для транслятора Ассемблера F.

Рассмотрим пример использования оператора ACTR.

Название	Операция	Операнды
MI &A E&A	MACRO	
	OPE	&P
	LCLA	&A
	ACTR	&P+1
	ANOP	
	SETA	&A+1
	EQU	&A
	AIF	(&A LE &P).MI
	MEND	

Данное макроопределение генерирует операторы EQU, количество которых задается параметром &P. Простым именам, которые называют эти операторы, присваиваются значения от 1 до значения параметра &P. Значением параметра &P может быть любой самоопределенный терм, поэтому используется оператор ASTR, разрешающий выполнять нужное количество переходов. При отсутствии этого оператора транслятор Ассемблера Е может сгенерировать не больше 150 операторов EQU; если бы значение &P было больше 150, транслятор реагировал бы на это, как на ошибку.

4.7. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ МАКРОСРЕДСТВ

4.7.1. Глобальные параметры

С помощью глобальных параметров можно передавать значения из одного макроопределения в другое или в основную часть программы. Переменный параметр должен быть определен как глобальный в каждом макроопределении (или в основной части программы), в котором он должен использоваться как глобальный параметр. Глобальные параметры могут быть арифметическими, логическими и знаковыми.

Первоначальные значения глобальным переменным параметрам определяют и присваивают операторы GBLA, GBLB и GBLC. Их формат:

Название	Операция	Операнды
Пробел	GBLA или GBLB или GBLC	Один или более переменных параметров, разделенных запятыми

Операторы GBLA, GBLB, GBLC могут встречаться как в макроопределении, так и в основной части программы. Однако первоначальные значения параметрам присваиваются только самым первым оператором, который встречается при генерации. Все последующие операторы GBLA, GBLB, GBLC, которые появляются в основной части программы или в других макроопределениях, не изменяют значение, присвоенное переменному глобальному параметру ранее. Если операторы GBLA, GBLB, GBLC используются в основной части программы, то они должны записываться после всех программных макроопределений, но перед всеми операторами основной части программы. За операторами GBLA, GBLB и GBLC должны следовать операторы LCLA, LCLB и LCLC, если они необходимы программе. Если команды определения глобальных переменных параметров используются в макроопределении, то они должны записываться сразу за оператором прототипа. Перед операторами определения переменных параметров и меж-

ду ними могут быть только операторы SPACE, TITLE, EJECT, ISEQ и PRINT. Значения глобальных переменных параметров, как и для локальных параметров, изменяются операторами SETA, SETB и SETC.

Рассмотрим пример использования трех макрокоманд: MK1, MK2 и MK3.

Название	Операция	Операнды	Идентификация
&A1	MACRO MK1	макроопределение MK1	1.
	GBLA LA	&A1	2
	MEND	2,&A1	3
	MACRO MK2	макроопределение MK2	4
	GBLA LA	&A1	5
	SETA LA	&A1+1	6
	MEND	2,&A1	7
	MACRO MK3	макроопределение MK3	8
	LCLA LA	&A1	9
	MEND	2,&A1	10
	MACRO MK3	макроопределение MK3	11
	LCLA LA	&A1	12
	MEND	2,&A1	13
	* основная часть программы		14
	GBLA LA	&A1	15
	SETA LA	1	16
&A1	MEND	2,&A1	17
	MACRO MK1	макрокоманда	18
	MACRO MK2	макрокоманда	19
	MACRO MK3	макрокоманда	20
	LCLA LA	2,&A1	21
	MEND		22
	END		23
			24
			25

В основной части программы и в макроопределениях MK1 и MK2 используется глобальный параметр &A1. В макроопределении MK3 нет глобального параметра, но определяется локальный параметр &A1. Устанавливаемые и используемые значения глобального параметра &A1 и локального параметра &A1 совершенно независимы. Первоначальное значение глобальному параметру &A1 устанавливает оператор 18, затем оператор 19 присваивает этому параметру значение, равное 1. Последнее используется в операторе 20. По макрокоманде MK1 в операторе 4 употребляется это же значение параметра &A1, равное 1. По макрокоманде MK2 оператор 9 увеличивает значение параметра &A1 на 1, поэтому в операторе 10 параметр &A1 заменяется значением 2. По макрокоманде MK3 в операторе 15 используется значение локального параметра, равное 0. В операторе 24 глобальный параметр заменяется значением, равным 2. В результате генерации будет получена следующая программа:

Название	Операция	Операнды	Идентификация
* основная часть программы			
	LA	2,1	20
	MK1	макрокоманда	21
	LA	2,1	
	MK2	макрокоманда	22
	LA	2,2	
	MK3	макрокоманда	23
	LA	2,0	
	LA	2,2	24

Глобальный параметр в нижеприведенном примере позволяет передавать значения из основной части программы в макроопределение.

Название	Операция	Операнды	Идентификация
&K	MACRO	макроопределение	1
	MK1	&P,&R	2
	GBLA	&N	3
	LCLC	&K	4
	SETC	'&P'(&N,1)	5
	AR	&R,&K	6
	MEND		7
* основная часть программы			8
&N	GBLA	&N	9
	SETA	2	10
&N	MK1	132,0 макрокоманда	11
	SETA	3	12
	MK1	132,0 макрокоманда	13
	END		14

Оператор 10 присваивает параметру &N значение, равное 2. Это значение используется при обработке макроопределения по первой макрокоманде, поэтому параметру &K присваивается знаковое значение 3. Для этой макрокоманды сгенерируется макрорасширение вида:

Название	Операция	Операнды
	AR	0,3

Оператором 12 глобальному параметру &N присваивается значение 3, поэтому для второй макрокоманды сгенерируется следующее макрорасширение:

Название	Операция	Операнды
	AR	Ø,2

4.7.2. Индексируемые переменные параметры

Переменный параметр считается индексируемым, если под его названием выступает несколько значений, каждое из которых можно использовать независимо от других. Эти значения могут быть арифметическими, знаковыми или логическими. Индексируемый переменный параметр записывается как обычный параметр, за которым сразу же следует индекс, заключенный в скобки.

Прежде чем использовать переменный параметр как индексируемый, его необходимо определить индексируемым в операторах GBLA, GBLB, GBLC, LCLA, LCLB или LCLC, т. е. (GBLX или LCLX). В командах определения переменных параметров индекс должен быть только десятичным самоопределенным термом. В остальных операторах индекс может быть арифметическим выражением. Во всех случаях значение индекса должно находиться в пределах от 1 до 255.

Если в операторе GBLX или LCLX переменный параметр определен как индексируемый, то в дальнейшем он должен использоваться только с индексом. И наоборот, переменный параметр, объявленный в операторе GBLX или LCLX неиндексируемым, нельзя использовать с индексом. Глобальный параметр, используемый в нескольких макроопределениях и (или) в основной части программы, должен всюду определяться только неиндексируемым или только индексируемым. При этом индекс у индексируемого параметра во всех операторах GBLX должен указываться один и тот же.

Индексируемые переменные параметры, которые связаны с одним именем объявленного параметра, получают те же начальные значения, что и переменные параметры соответствующего типа без индекса. Рассмотрим пример использования индексируемого переменного параметра.

Название	Операция	Операнды
.REP &I &L(&I)	MACRO NEW LCLA LCLC ANOP SETA SETC AIF DC MEND	&I &L(4) &I+1 '&SYSLIST(&I)'(2,1) (&I LE 4).REP C'&L(1).&L(2).&L(3).&L(4)'

Макроопределение строит константу, состоящую из вторых знаков первых четырех операндов макрокоманды, с помощью индексированного переменного параметра &L. Размерность его равна 4. Это значит, что используются четыре значения знакового параметра, связанные с одним именем. В операторе SETC в качестве индекса переменного параметра &L употребляется арифметическое значение, задаваемое арифметическим параметром &I. Оператор SETA присваивает параметру &I значения от 1 до 4. К очередному операнду макрокоманды обращаются с помощью системного параметра &SYSLIST(&I). Оператор AIF позволяет обработать первые четыре операнда и присвоить нужные значения каждому индексу параметра &L. Присвоение выполняется одним оператором SETC, потому что арифметический параметр &I, являющийся индексом параметра &L, принимает значение от 1 до 4.

В рассматриваемом примере использование индексированного параметра сокращает длину макроопределения, потому что один оператор SETC присваивает несколько знаковых значений индексам переменного параметра. Если бы не было индексированных параметров, данное макроопределение выглядело бы так:

Название	Операция	Операнды
	MACRO	
	NEW	
	LCLC	&L1,&L2,&L3,&L4
&L1	SETC	'&SYSLIST(1)')(2,1)
&L2	SETC	'&SYSLIST(2)')(2,1)
&L3	SETC	'&SYSLIST(3)')(2,1)
&L4	SETC	'&SYSLIST(4)')(2,1)
	DC	C'&L1.&L2.&L3.&L4'
	MEND	

В этом макроопределении пришлось употребить несколько операторов SETC, потому что в генераторном цикле без индексированного параметра невозможно обращаться к разным независимым значениям с помощью одного названия параметра. В то же время индексированные параметры можно применять и в обычных случаях, наравне с неиндексированными параметрами, например:

Название	Операция	Операнды
	MACRO	
	DI	
	LCLA	&P1,&P2,&P3
	LCLC	&A(2)
&A(1)	SETA	&C
&A(2)	SETA	&P1
&C	SETC	&P2
	DC	'&P3'(&A(1),&A(2))
	MEND	C'&C'

В приведенном примере без ущерба качества макроопределения вместо индексированного параметра можно использовать два неиндексированных.

4.7.3. Оператор MNOTE

Оператор MNOTE дает возможность сообщить программисту о том, как проходит обработка макроопределения (например, в макрокоманде есть неправильные операнды). Транслятор выдает распечатку всей транслируемой программы, при этом для макрокоманд печатаются операторы макрорасширения, а среди них — сообщение оператора MNOTE. Данные о нем печатаются также в списке сообщений об ошибках в программе. Формат оператора MNOTE:

Название	Операция	Операнды
Имя перехода или пробел	MNOTE	Два операнда, разделенные запятой

Первым операндом может быть любое десятичное число от 0 до 255 и знак *. Этот операнд может быть использован в качестве кода ошибки. Код ошибки можно выбирать произвольно, на обработку макроопределения он не влияет. Код ошибки печатается вместе с сообщением оператора MNOTE. Если кодом ошибки является знак *, то сообщение об операторе MNOTE не появляется в списке сообщений о всех ошибках в программе. Поэтому знак * для обозначения кода ошибки следует употреблять только при выдаче информационного сообщения о генерации. Если необходимо сообщить об ошибке генерации, то в качестве кода ошибки нужно использовать десятичное число.

Вторым операндом оператора MNOTE записывается текст сообщения, который должен заключаться в апострофы. Если в этом тексте нужно использовать апостроф или знак &, то они должны записываться соответственно как два апострофа или два знака &. При обработке оператора MNOTE в текст программы будет генерироваться содержимое поля операндов оператора MNOTE. Апострофы, в которые заключен текст сообщения, при этом опускаются. Если в тексте сообщения записан параметр, то он заменяется знаками, соответствующими этому параметру.

В нижеследующем примере рассматривается использование оператора MNOTE. Переход на оператор 10 происходит в том случае, если неправильно записано значение параметра &R1. Оператор 10 сгенерирует сообщение о неправильной записи операнда, соответствующего параметру &R1. Переход на оператор 12, который генерирует сообщение о неправильной записи значения параметра &R2, происходит в том случае, если неправильно записан второй операнд макрокоманды. Если операнды макрокоманды

записаны верно, то оператору 7 сгенерируется сообщение, сигнализирующее о правильной записи операндов. Однако это сообщение не появится в списке сообщений о всех ошибках в программе в отличие от сообщений операторов 10 и 12.

Название	Операция	Операнды	Идентификация
	MACRO		1
	SUM	&R1,&R2	2
	AIF	(T'&R1 NE 'N') .M1	3
	AIF	(T'&R2 NE 'N') .M2	4
	AIF	(&R1 GT 15) .M1	5
	AIF	(&R2 GT 15) .M2	6
	MNOTE	*, 'REAL OPERANDS'	7
	AR	&R1,&R2	8
	AGO	.MEND	9
.M1	MNOTE	1Ø, 'ILLEGAL OPERAND 1—&R1'	1Ø
	AGO	.MEND	11
.M2	MNOTE	1Ø, 'ILLEGAL OPERAND 2—&R2'	12
.MEND	MEND		13

В макрокоманде, приведенной ниже, записаны правильные операнды:

Название	Операция	Операнды
	SUM	2,X'B'

Для такой макрокоманды сгенерируется соответствующее макрорасширение:

Название	Операция	Операнды
	AR	*, 'REAL OPERANDS 2,X'B'

В следующей макрокоманде записано неправильное значение параметра &R1:

Название	Операция	Операнды
	SUM	27,X'A'

Для этой макрокоманды сгенерируется другое макрорасширение, в которое входит только сообщение о неправильной записи операнда.

Название	Операция	Операнды
		1Ø, ILLEGAL OPERAND 1—27

В нижепредставленной макрокоманде неправильно записано значение второго операнда:

Название	Операция	Операнды
	SUM	2, =F'1'

Для этой макрокоманды сгенерируется следующее макрорасширение:

Название	Операция	Операнды
		1Ø, ILLEGAL OPERAND 2—=F'1'

4.7.4. Оператор MEXIT

По оператору MEXIT транслятор прекращает обработку данного макроопределения. Следующим обрабатывается оператор, который стоит сразу же за макрокомандой, соответствующей данному макроопределению. Формат оператора MEXIT:

Название	Операция	Операнды
Имя перехода или пробел	MEXIT	Не используется

Рассмотрим макроопределение, в котором используется данный оператор:

Название	Операция	Операнды
	MACRO	
	ADD2	&P1,&P2
	AIF	(T'&P2(1) NE 'F' OR T'&P2(2) NE 'F').A1
	AIF	(T'&P1 NE 'N').A2
	AIF	(&P1 GE 16).A2
	L	&P1,&P2(1)
	A	&P1,&P2(2)
	ST	&P1,&P2(1)
.A1	MEXIT	
	MNOTE	55,'OPERAND &P2 IS ERROR'
.A2	MEXIT	
	MNOTE	55,'OPERAND &P1 IS ERROR'
	MEND	

Это макроопределение выполняет сложение двух чисел. Символические имена областей, в которых находятся суммируемые числа, задаются списком операндов, соответствующим параметру &P2. Первый оператор AIF проверяет, является ли буква F характеристикой типа обоих элементов списка. Если характеристика типа буква F, то проверяется параметр &P1, который задает номер общего регистра. Его значение должно быть в пределах от 0 до 15. При выполнении этого условия генерируются команды, подсчитывающие сумму чисел и запоминающие ее. Если операнды в макрокоманде записаны неправильно, то оператором MNOTE генерируется сообщение об ошибке, после чего оператор MEXIT прекращает обработку макроопределения.

4.8. ПРИМЕРЫ СОСТАВЛЕНИЯ МАКРООПРЕДЕЛЕНИЙ

Рассматриваемые примеры практического значения не имеют, они только иллюстрируют возможности макросредств.

4.8.1. Извлечение корня

Составить макроопределение, которое извлекает квадратный корень из числа с плавающей точкой типа D, т. е. вычисляет значение функции $y = \sqrt{x}$. Вычисления выполняются по итерационной формуле Ньютона

$$y_{n+1} = \frac{1}{2} \left(y_n + \frac{x}{y_n} \right),$$

где x — число, из которого извлекается корень;

y_n — текущее приближение.

Значением квадратного корня считается y_{n+1} , если $|y_{n+1} - y_n| < \varepsilon$, где ε — абсолютная погрешность. В качестве операндов макрокоманды используются:

x — аргумент функции (соответствует параметру &ARG);

ε — абсолютная погрешность (соответствует параметру &POG).

Значения операндов могут задаваться литералами или простыми именами областей памяти, в которых находятся аргумент и погрешность. Вычисленное значение квадратного корня должно размещаться в регистре с плавающей точкой, с номером 0.

Текст макроопределения:

Название	Операция	Операнды	Идентификация
	MACRO		1
	KOP	&ARG,&POG	2
	LD	6,&ARG	3
	LTDR	6,6	4
	BP	L&SYSNDX	5
	SDR	0,0	6
	B	M&SYSNDX+L'M&SYSNDX	7

Название	Операция	Операнды	Идентификация
L&SYSNDX K&SYSNDX	LD	$4, = D'1'$	8
	LDR	$\emptyset, 6$	9
	DDR	$\emptyset, 4$	10
	ADR	$\emptyset, 4$	11
	MD	$\emptyset, = D'.5'$	12
	LDR	$2, \emptyset$	13
	SDR	$2, 4$	14
	LPDR	$2, 2$	15
	LDR	$4, \emptyset$	16
	CD	$2, \&POG$	17
M&SYSNDX	BH	K&SYSNDX	18
	MEND		19

Операторами макроопределения в данном примере являются только машинные команды. Все машинные команды будут генерироваться в макрорасширение. При обработке макроопределения постоянные параметры заменятся значениями, записанными в макрокоманде.

При выполнении программы каждая машинная команда макрорасширения реализует определенное действие. Операторы 3, 4 и 5 проверяют знак аргумента. Если аргумент отрицательный, то результатом считается 0 и с помощью машинной команды BP происходит выход из макрорасширения в основную часть программы. Этот выход при генерации не вызывает окончания обработки макроопределения, просто при выполнении команд, которые генерируются из макроопределения, происходит переход на команду, размещаемую сразу за командами макроопределения. Если аргумент положительный, то при выполнении программы происходит переход на оператор 8, который начинает вычисление значения квадратного корня. В регистр 6 загружается аргумент функции, а в регистр 4 — число 1, которое принимается за первое приближение — y_1 . Оператор 9 начинает вычисление следующего приближения — y_{n+1} . Оператор 13 начинает проверку вычисленного приближения. Если разность между текущим приближением y_n и вычисленным приближением y_{n+1} меньше погрешности, вычисления прекращаются. В противном случае значение y_{n+1} принимается за текущее приближение и повторяется вычисление следующего приближения.

В данном макроопределении корень вычисляется с помощью машинных команд. Такие вычисления нельзя реализовать операторами SETA, SETC, SETB, AIF и AGO, потому что они позволяют вычислять и проверять только значения параметров. Но параметры используются лишь при трансляции программы транслятором для замены их соответствующими им значениями.

Значение функции можно вычислить только с помощью машинных команд при выполнении протранслированной программы.

В любом случае все вычисления, которые происходят во время выполнения протранслированной программы, нужно реализовывать только машинными командами.

Макроопределение, составленное для данного примера, используется в следующей программе:

Название	Операция	Операнды		Идентификация
ST	START	X'3000'		1
	BALR	11,0		2
	USING	*,11		3
	KOP	ARG1,=D'2E-8'		4
	KOP	ARG2,=F'7'		5
	SVC	14		6
ARG1	DC	D'22'		7
ARG2	DC	E'31'		8
	END	ST		9

Макроопределение KOP применяется дважды. Для первой макрокоманды (оператор 4) сгенерируется правильное макрорасширение, потому что операнды макрокоманды определяют правильные числа с плавающей точкой. Для второй макрокоманды (оператор 5) сгенерируется макрорасширение, команды которого будут работать неправильно, потому что в этой макрокоманде неверно использованы исходные данные: число с плавающей точкой типа E и число с фиксированной точкой типа F. Команды же макроопределения обрабатывают числа с плавающей точкой типа D, поэтому при выполнении протранслированной программы получатся неточные результаты. Можно было бы вставить в макроопределение проверку правильности входных данных, тогда для второй макрокоманды выдалось бы сообщение об ошибке, а макрорасширение не сгенерировалось.

4.8.2. Генерация констант типа C

Составить макроопределение, которое генерирует константы типа C. Значением любой константы должны быть первые три знака каждого операнда макрокоманды. Количество операндов в разных макрокомандах может быть различным. Генерировать константу для операнда нужно только в том случае, если среди знаков этого операнда нет цифр.

В нижеприведенном макроопределении (его блок-схема представлена на рис. 7) арифметический параметр &N используется в качестве счетчика операндов макрокоманды, параметр &A — знаков операнда. Операнды обрабатываются с помощью двух генераторных циклов. Внешний цикл просматривает все операнды макрокоманды, внутренний — знаки одного операнда.

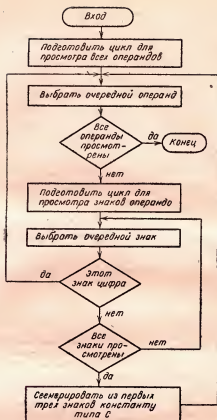


Рис. 7. Генерация констант типа C

Название	Операция	Операнды
.N1 &N	MACRO	
	DCC	
	LCLC	&K
	LCLA	&N,&A
	ANOP	
&A	SETA	&N+1
	AIF	(&N GT N'&SYSLIST).MEND
	SETA	Ø
.A1 &A	ANOP	
	SETA	&A+1
	AIF	('&SYSLIST(&N)'(&A,1) LT 'Ø').A2
A2 &K	AIF	('&SYSLIST(&N)'(&A,1) LE '9').N1
	AIF	(&A LT K'&SYSLIST(&N)).A1
	SETC	'&SYSLIST(&N)'(1,3)
.MEND	DC	C'&K'
	AGO	.N1
	MEND	

Первый оператор AIF проверяет, все ли операнды обработаны. Запись N'&SYSLIST используется для определения количества операндов макрокоманды. Второй и третий оператор AIF устанавливают, является ли очередной знак операнда цифрой: если цифра, то обрабатывается следующий операнд макрокоманды. Последний оператор AIF проверяет количество обработанных знаков операнда. После обработки всех знаков параметру &K присваивается значение первых трех знаков операнда, а потом генерируется оператор DC.

После построения очередной константы оператор AGO передает управление на обработку следующего операнда. Предположим, обрабатывается макрокоманда:

Название	Операция	Операнды
	DCC	LMN,A"U",AIB,F(E=G)

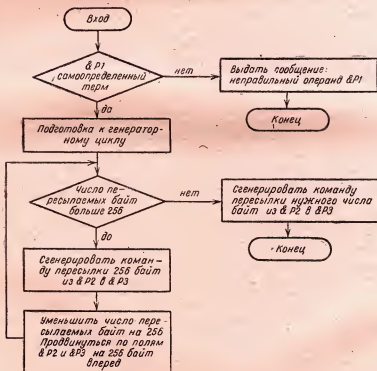


Рис. 8. Пересылка содержимого области памяти

По ней строятся такие операторы:

Название	Операция	Операнды
	DC	C'LMN'
	DC	C'A'''
	DC	C'F(E'

Для каждого операнда макрокоманды сгенерирован один оператор DC. Среди знаков третьего операнда встречается цифра 1, поэтому оператор DC для него генерироваться не будет.

4.8.3. Пересылка содержимого области памяти

Составить макроопределение, которое пересылает определенное количество байт из одной области памяти в другую. Количество пересылаемых байт задается первым постоянным параметром &P1. Параметр &P2 именует область памяти, содержимое которой пересылается в другую область, а параметр &P3 — область, в которую пересылается содержимое области, заданной параметром &P2.

Блок-схема макроопределения представлена на рис. 8, а текст макроопределения приведен ниже.

Название	Операция	Операнды	Идентификация
	MACRO		1
	PERES	&P1,&P2,&P3	2
	LCLA	&A1,&A2	3
	AIF	(T'&P1 NE 'N').M1	4
&A1	SETA	&P1	5
.AIF	AIF	(&A1 LE 256).M2	6
	MVC	&P3+&A2.(256),&P2+&A2	7
&A1	SETA	&A1-256	8
&A2	SETA	&A2+256	9
	AGO	.AIF	10
.M2	MVC	&P3+&A2.(&A1),&P2+&A2	11
	MEXIT		12
.M1	MNOTE	15,' неправильный операнд &P1'	13
	MEND		14

Оператор 4 проверяет, правильно ли записано в макрокоманде значение параметра &P1. Если значение неправильное, об этом с помощью оператора 13 выдается сообщение. Если значение правильное, то оператор 5 устанавливает арифметическому параметру &A1 значение, равное значению параметра &P1 (количество пересылаемых байт).

Оператор 6 анализирует количество пересылаемых байт, т. е. проверяет, можно все байты переслать одной командой MVC или потребуются несколько таких команд. Если число пересылаемых

байт больше 256, то генерируется команда 7, которая пересылает 256 байт. Адрес области, в которую нужно пересылать байты, задается выражением $\&P3 + \&A2$. Первоначальное значение арифметического параметра $\&A2$ равно нулю, поэтому первая команда задает значение $\&P2 + 0$, т. е. начало области, в которую должны пересылаться байты. Аналогично записано выражение, задающее область, из которой должны пересылаться байты. Оператор 8 уменьшает количество пересылаемых байт, потому что 256 байт уже перенесено в другую область. Оператор 9 увеличивает значение $\&A2$ на 256, тем самым устанавливая адрес области для следующей команды MVC. Если будет сгенерирована новая команда MVC, то она будет пересылать следующие 256 байт в другое место, после чего оператор 6 вновь проверит количество байт, оставшихся для пересылки.

Если количество пересылаемых байт не больше 256, то генерируется оператор 11, который пересылает столько байт, сколько определено значением параметра $\&A1$.

В операторах 7 и 11 указывается явная длина операндов, которая записывается в скобках за параметром $\&A2$. Для того чтобы выражение, данное в скобках, при генерации соединилось со значением параметра $\&A2$, за последним ставится точка. Если бы точки не было, транслятор считал длину, записанную за параметром, индексом этого параметра.

В нижеприведенной макрокоманде значение параметра $\&P1$ меньше 256.

Название	Операция	Операнды
	PERES	211,N,K

Для такой макрокоманды сгенерируется макрорасширение:

Название	Операция	Операнды
	MVC	$K + \emptyset(211), N + \emptyset$

В следующей макрокоманде значением параметра $\&P1$ является число X'ZAB' больше 256.

Название	Операция	Операнды
	PERES	X'ZAB',N,K

Для такой макрокоманды сгенерируется макрорасширение, состоящее из четырех команд MVC.

Название	Операция	Операнды
	MVC	$K + \emptyset (256), N + \emptyset$
	MVC	$K + 256 (256), N + 256$
	MVC	$K + 512 (256), N + 512$
	MVC	$K + 768 (171), N + 768$

4.8.4. Генерация знаков констант в обратном порядке

Составить макроопределение, позволяющее создать константу типа С для тех операндов, у которых вторым знаком является буква L. Количество знаков в операнде должно быть не больше восьми. Количество операндов в макрокоманде может быть переменным, операнды макрокоманды обрабатываются в обратном порядке. Знаки операндов в константе также располагаются в обратном порядке, т. е. первым знаком в константе должен быть последний знак операнда, вторым знаком константы — предпоследний знак операнда и т. д.

Блок-схема макроопределения представлена на рис. 9, а текст макроопределения приведен ниже.

Название	Операция	Операнды	Идентификация
	MACRO		1
	MADC1		2
	LCLA	&N,&K	3
	LCLC	&P(8)	4
&N	SETA	N'&SYSLIST+1	5
.ANOP1	ANOP		6
&K	SETA	\emptyset	7
.AN	ANOP		8
&K	SETA	$\&K + 1$	9
&P(&K)	SETC	' '	10
	AIF	(&K LE 7).AN	11
&N	SETA	$\&N - 1$	12
&K	SETA	K'&SYSLIST(&N)	13
	AIF	(&N NE \emptyset).M	14
	MEXIT		15
.M	AIF	(&K LT 2 OR &K GT 8).ANOP1	16
	AIF	('&SYSLIST(&N)'(2,1) NE 'L').ANOP1	17
.ANOP2	ANOP		18
&P(&K)	SETC	'&SYSLIST(&N)'(&K,1)	19
&K	SETA	$\&K - 1$	20
	AIF	(&K NE \emptyset).ANOP2	21
	DC	C'&P(8)&P(7)&P(6)&P(5)&P(4)&P(3)&P(2)&P(1)'	22
	AGO	.ANOP1	23
	MEND		24

Операторы 6—11 присваивают пустое знаковое значение всем параметрам, связанным с именем &P. После этого проверяется, все ли операнды макрокоманды просмотрены. Если они просмотрены, то оператор 22 устанавливает флаг X.

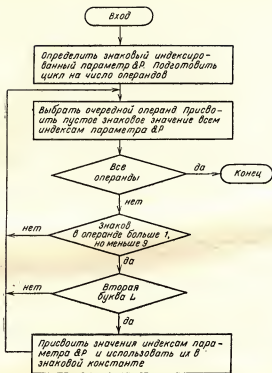


Рис. 9. Генерация знаков констант в обратном порядке

рены, обработка макроопределения прекращается, в противном случае операторами 16 и 17 проверяется очередной операнд макрокоманды. Если число знаков в операнде меньше двух или больше восьми или вторая буква операнда не является буквой L, то проверяется очередной операнд макрокоманды. Если для данного операнда должна создаваться константа, то операторы 19, 20 и 21 присваивают нужные значения знаковым параметрам, связанным с именем &P. Каждому параметру в качестве значения присваивается один знак операнда: параметру &P(1) соответствует первый знак операнда, параметру &P(2) — второй знак и т. д.

Количество повторений цикла, который присваивает знаковым параметрам знаковые значения, равно количеству знаков в операнде. Поэтому если число знаков в операнде меньше восьми, то отдельным параметрам, связанным с именем &P, будут соответствовать пустые знаковые значения.

Оператор 22 определяет константу типа C. Знаки ее задаются знаковыми параметрами, связанными с именем &P. Эти параметры в операторе 13 перечисляются в обратном порядке, поэтому знаки в константе будут расположены также в обратном порядке.

ПРОГРАММИРОВАНИЕ ВВОДА — ВЫВОДА

Операционные системы, созданные для ЕС ЭВМ, предоставляют программисту возможность пользоваться стандартными процедурами ввода — вывода. Эти процедуры реализованы в виде макроопределений на языке Ассемблера.

В настоящей главе кратко рассматриваются только основные принципы организации ввода — вывода на языке Ассемблера в операционной системе ДОС ЕС, так как данный вопрос требует отдельного тщательного изучения.

5.1. СИСТЕМА ВВОДА — ВЫВОДА

В ДОС ЕС функции управления вводом — выводом выполняет специальная система, которая предоставляет программисту стандартные процедуры для управления данными и обеспечения доступа к ним. Система различает два уровня управления данными: физический и логический. Физический уровень предполагает точное знание внешних устройств. Логический уровень освобождает программиста от необходимости точно знать конкретные внешние устройства.

Каждый уровень для программиста, использующего язык Ассемблера, реализуется совокупностью макроопределений, составляющих соответственно физическую систему управления вводом — выводом и логическую систему управления вводом — выводом.

5.1.1. Физический уровень управления
вводом — выводом

При организации ввода — вывода на физическом уровне программист сам подготавливает канальную программу, содержащую команды ввода — вывода, которые необходимо выполнить для данного конкретного внешнего устройства.

Для определения команд ввода — вывода в канальной программе можно воспользоваться оператором с мнемоническим кодом операции CCW. По команде Ассемблера CCW (определить команду ввода — вывода) определяется команда ввода — вывода, состоящая из 8 байт. Эта команда размещается на границе двойного слова. Байты, пропущенные при выравнивании, заполняются нулями.

Формат оператора CCW:

Название	Операция	Операнды
Любое символическое имя или пробел	CCW	Четыре операнда, разделенные запятыми

Все операнды в операторе CCW должны присутствовать и следовать в определенном порядке. Они представляют собой следующее:

операнд 1 — абсолютное выражение, определяющее код команды. Его значение помещается в байте 1 поля, предназначенного для команды ввода — вывода;

операнд 2 — абсолютное или переместимое выражение, определяющее адрес данных. Значение этого выражения помещается в байтах 2—4;

операнд 3 — абсолютное выражение, определяющее флажки команды ввода — вывода. Его значение располагается в байте 5. Байт 6 команды ввода — вывода устанавливается в нуль;

операнд 4 — абсолютное выражение, определяющее число байт для операции. Значение этого выражения помещается в байтах 7—8.

Для выполнения канальной программы организуется обращение к супервизору. При этом программист, кроме канальной программы, подготавливает специальную информацию, называемую блоком управления данными. В блоке указываются адрес канальной программы в основной памяти, информация о действиях в случае возникновения ошибок и т. д.

При программировании ввода — вывода на физическом уровне программист может воспользоваться макрокомандами физического уровня, например CCB (создать блок управления данными), EXCP (выполнить канальную программу), WAIT (ожидать завершения операции ввода — вывода) и др. Однако и в этом случае программирование ввода — вывода на физическом уровне — довольно трудоемкий процесс, поэтому в проблемных программах ввод — вывод чаще всего организуется на логическом уровне.

5.1.2. Логический уровень управления вводом — выводом

В отличие от физического уровня управления вводом — выводом логический уровень обрабатывает данные исходя из их логического содержания. Здесь существенными являются формат и организация данных, важны такие понятия, как файл (совокупность записей, объединенных по некоторому общему признаку или признакам), логическая запись (логическая единица информации). Одна или несколько логических записей могут быть включены в физическую запись. Физическая запись называется также

блоком данных. Средства логического уровня ввода — вывода выполняют следующие функции:

- обращение к физическому уровню ввода — вывода с указанием выполнения соответствующих канальных программ;
- объединение в блок и выделение записей в блоке;
- переключение между областями ввода — вывода, если для работы с файлом используются две области;
- обработку состояния КОНЕЦ ФАЙЛА и КОНЕЦ ТОМА;
- проверку и запись меток файлов и др.

5.1.3. Организация данных

Как уже отмечалось, на логическом уровне управления вводом — выводом используются логические записи, объединенные в файлы. В каждом конкретном случае файл данных характеризуется размером, который он занимает на внешнем носителе, форматом записей, из которых он состоит, принципом следования в нем записей и т. д. Файл может содержать записи одного из трех форматов: фиксированной длины, переменной длины и неопределенной длины. Название формата определяется длиной записей, из которых состоит файл. Для некоторых типов устройств допускаются записи любого из указанных форматов (магнитная лента, диски), для других — только фиксированной длины (например, для перфокарточных устройств).

Записи в файлах на магнитных лентах и дисках могут быть сгруппированы в блоки. По этому признаку различаются записи блокированные и неблокированные. Блокирование записей сокращает число операций ввода — вывода, требуемых для обработки файла, и экономит внешнюю память, так как уменьшает количество промежутков между записями.

Файлы на магнитных лентах и дисках могут иметь метки. Их информация используется программами логической системы управления вводом — выводом для поиска требуемого файла, для защиты файла в системе от недозволенного вмешательства, а также для организации работы с этим файлом.

В зависимости от того, какие функции выполняются над данными файла, файлы подразделяются на вводные, выводные и рабочие. Вводные или выводные файлы могут применяться соответственно для ввода или вывода информации в пределах одного открытия и закрытия файла. Рабочий файл может быть использован как для ввода, так и для вывода информации в пределах одного открытия и закрытия файла.

5.2. ОРГАНИЗАЦИЯ ВВОДА — ВЫВОДА НА ЯЗЫКЕ АССЕМБЛЕРА

При записи программы на языке Ассемблера программисту для организации операций ввода — вывода лучше всего воспользоваться возможностями, которые предоставляет ему логический

уровень управления вводом — выводом. В этом случае программист освобождается от подготовки в своей программе канальных программ и другой информации, требующей знания внешних устройств и команд ввода — вывода для этих устройств. Все функции логического уровня управления вводом — выводом программист может использовать с помощью макрокоманд ввода — вывода. Макроопределения, соответствующие макрокомандам ввода — вывода, записаны на языке Ассемблера и хранятся в библиотеке исходных модулей.

Макрокоманды могут быть двух видов: декларативные и императивные. Декларативные (описательные) макрокоманды служат для создания таблиц, содержащих характеристики файлов. Операционная система требует, чтобы все подлежащие обработке файлы были описаны. Таблицы, описывающие файлы, используются программами логического уровня управления вводом — выводом. Императивные (повелительные) макрокоманды указывают действия, которые должны быть выполнены в процессе обработки. Такими действиями могут быть: открытие файла, считывание блока с внешнего носителя в основную память и т. д.

Декларативной макрокомандой для описания файлов является макрокоманда DTFXX (XX — позиции букв, указывающих конкретную макрокоманду DTF). Для определения файлов, обрабатываемых программами логического уровня управления вводом — выводом, используются макрокоманды описания файлов, ориентированные на устройства, например:

DTFCD — для файлов на устройствах ввода и вывода перфокарт;

DTFMT — для файлов на магнитной ленте;

DTFPR — для файлов на устройстве печати.

Следовательно, если в программе должен выполняться ввод или вывод данных, программист прежде всего должен написать в программе макрокоманду DTFXX соответствующую тому внешнему устройству, с которого будут вводиться данные или на которое они будут выводиться.

Каждая макрокоманда DTFXX должна согласовываться с логическим модулем ввода — вывода. Логический модуль — это программа, обеспечивающая выполнение необходимых функций ввода — вывода, требуемых в проблемной программе при обработке логического файла. Например, модуль может читать и писать данные, проверять необычные состояния ввода — вывода, при необходимости объединять в блок и выделять из блока логические записи, помещать логические записи в рабочую область.

Логические модули включаются в программу при трансляции по декларативной макрокоманде XXMOD (XX — позиции букв, указывающих необходимый логический модуль). Каждый модуль должен согласовываться с соответствующей макрокомандой DTFXX. Например, макрокоманде DTFMT (определяет файл на магнитной ленте) должна соответствовать макрокоманда MTMOD

(генерирует логический модуль для работы с файлом на магнитной ленте).

Таким образом, определив в программе с помощью макрокоманд DTFXX необходимый файл данных, программист должен еще сгенерировать в этой программе логический модуль для работы с данным файлом с помощью соответствующей макрокоманды XXMOD. Включение в модуль тех или иных функций при его генерации производится на основании параметров, указываемых в макрокоманде XXMOD. Программист может выбрать лишь те функции, которые необходимы его программе.

Записав в программе макрокоманды DTFXX и XXMOD, программист тем самым подготовил все для выполнения операций ввода — вывода над файлом данных: описал имеющийся файл и составил программу, которая может выполнить операции ввода — вывода на этом файле. Если необходимо выполнить в программе операцию ввода — вывода (например, чтение) программист должен записать императивную макрокоманду ввода — вывода, соответствующую необходимой операции. Например, можно использовать следующие группы императивных макрокоманд:

макрокоманды подготовки файла к использованию (например, OPEN);

макрокоманды обработки, позволяющие выводить и вводить отдельные записи файла (например, PUT, GET);

макрокоманды завершения работы с файлом (например, CLOSE).

Рассмотрим простейший пример организации ввода — вывода в программе на языке Ассемблера.

Имеется массив перфокарт с информацией в коде ДКОИ. Необходимо распечатать содержание каждой перфокарты.

Программа, выполняющая поставленную задачу, будет выглядеть следующим образом:

Название	Операция	Операнды	Идентификация
INKART	DTFCD	DEVADDR=SYSIPT,	1
		IOAREAI=OBL, имя области ввода	2
		BLKSIZE=80, длина области ввода	3
		DEVICE=6012,	4
		EOFADDR=KOH, п/п обработки конца	5
		MODNAME=INK, имя модуля ввода	6
		RECFORM=FIXUNB, формат записей	7
		TYPEFLE=INPUT, тип файла	8
			9
OUTPRIN	DTFPR	DEVADDR=SYSLST,	10
		IOAREAI=OBL,	11
		BLKSIZE=120,	12
		DEVICE=7035,	13
		MODNAME=OUTPR,	14
		CONTROL=YES,	15
		RECFORM=FIXUNB	16
INK	CDMOD	DEVICE=6012	17

Название	Операция	Операнды	Идентификация
OUTPR BEGIN	PRMOD	CONTROL=YES, RECFORM=FIXUNB	18
	BALR	11,0	19
	USING	*,11	20
POVT	OPEN	INKART, OUTPRIN	21
	GET	INKART	22
	CNTRL	OUTPRIN, SP, 1	23
	PUT	OUTPRIN	24
	B	POVT	25
KON	CLOSE	INKART, OUTPRIN	26
	EOJ		27
	DS	0D	28
OBL	DS	80C	29
	DC	40C	30
	END	BEGIN	31

В программе прежде всего описываются два файла данных: файл на перфокартах, который должен вводиться, и выводной файл на печати, который должен содержать распечатку информации на перфокартах.

Вводной файл на перфокартах описывается макрокомандой DTFCД с именем INKART. Имя в поле названия макрокоманды является символическим именем файла. При записи макрокоманды используются строки продолжения, вся информация для макрокоманды занимает строки 2—9. С помощью ключевых параметров (каждый параметр записан в отдельной строке) в макрокоманде приводятся данные о файле: с какого логического устройства его необходимо вводить (DEVADDR=SYSIPT), имя области, куда следует вводить записи файла (IOAREA1=OBL), формат записей, составляющих файл (RECFORM=FIXUNB, записи фиксированной длины), имя подпрограммы для обработки состояния КОНЕЦ ФАЙЛА (EOFADDR=KON), шифр физического устройства, с которого вводится файл (DEVICE=6012). Тип файла описывается с помощью параметра TYPEFLE=INPUT. Параметр MODNAME=INK указывает имя логического модуля, выполняющего ввод перфокарт.

Выводной файл на печати описывается макрокомандой DTFRP с именем OUTPRIN. Макрокоманда имеет несколько строк продолжения (строки 11—16). Информация о файле, данная с помощью параметров DEVADDR, IOAREA1, BLKSIZE, DEVICE, RECFORM, аналогична информации в макрокоманде DTFCД. Параметр CONTROL=YES указывает, что при выводе на печать будет использоваться для управления выводом макрокоманда CNTRL. Параметр MODNAME=OUTPR определяет имя логического модуля для вывода на печать.

После макрокоманд описания файлов, необходимых для работы, в программе записаны макрокоманды CDMOD и PRMOD, по

которым в программу будут включены логические модули, т. е. программы, выполняющие операции ввода — вывода. Имена этих макрокоманд являются именами логических модулей и указываются в соответствующих макрокомандах DTFCД и DTFPR.

Записывая другие параметры в макрокомандах DTF и MOD, можно предоставлять иное описание файлов или вызывать построение различных модулей. Например, при описании вводного файла в макрокоманде DTFCД указывается имя одной области ввода. Параметры, записанные в макрокоманде CDMOD, вызовут построение модуля, который выполняет ввод в одну область. Можно было в макрокоманде CDMOD записать еще параметр IOAREA2=YES, что вызвало бы построение модуля, выполняющего ввод в две области ввода.

После того как с помощью описательных макрокоманд DTF и MOD подготовлена возможность выполнения операций ввода — вывода, в программе записываются операторы, осуществляющие поставленную задачу. Здесь для ввода — вывода используются повелительные макрокоманды.

Операторы 19 и 20 определяют регистр базы для программы. Оператор 21 — это макрокоманда OPEN, выполняющая открытие файлов данных. В качестве параметров называются имена файлов, которые должны быть открыты: INKART, OUTPRIN. Далее записывается макрокоманда GET (оператор 22), которая делает доступной для обработки в области ввода очередную логическую запись вводного файла (будет вводиться одна перфокарта). Имя вводного файла указывается операндом макрокоманды. Ввод выполняется в область ввода, адрес которой указан при описании файла макрокомандой DTFCД, т. е. в область с именем OBL.

Макрокоманда CNTRL применяется для выполнения операций без передачи физических данных. В программе она используется для управления выводом распечатки. В макрокоманде в качестве операндов указаны: имя файла — OUTPRIN, мнемонический управляющий код необходимой операции — SP (пропуск при печати 1, 2 или 3 строк), число пропускаемых строк — 1. По макрокоманде CNTRL перед выводом на печать строки информации будет пропущена одна строка.

Макрокоманда PUT (оператор 24) выполняет вывод логической записи, которая построена в области вывода. Имя выводного файла (OUTPRIN) указывается операндом макрокоманды. Адрес области вывода (OBL) определяется при описании файла OUTPRIN. Адрес области вывода совпадает с адресом области ввода. Это значит, что на печать выводится информация с перфокарты. В макрокоманде DTFPR длина области вывода определена в 120 байт (по числу позиций в строке на печати), поэтому после области OBL, состоящей из 80 байт (она заполняется при вводе), резервируется область из 40 пробелов. Таким образом, макрокоманда PUT выполнит вывод на печать информации с одной карты. Оператор 25 передает управление на выполнение ввода и распечатку очередной карты. Он выполняет безусловный переход.

Возникает вопрос, когда и каким образом будет закончен ввод перфокарт.

Вводной файл на перфокартах должен заканчиваться признаком конца файла /*. При вводе такой карты логический модуль, выполняющий ввод карт, передает управление по адресу, указанному при описании вводного файла параметром EOFADDR. В программе этот адрес указан символическим именем КОН. Таким образом, по концу вводного файла будет выполнен переход из логического модуля по адресу КОН. По этому адресу в программе находится макрокоманда CLOSE (оператор 26), выполняющая закрытие файлов. Имена закрываемых файлов указаны операндами макрокоманды.

После закрытия файлов выполнение программы завершается с помощью макрокоманды EOJ (оператор 27).

Макрокоманды DTFCD, DTFPR, CDMOD, PRMOD записаны в начале программы, но это не обязательно. Они могут находиться и в другом месте программы, например, перед оператором END.

Из рассмотренного примера видно, что ввод — вывод на языке Ассемблера может быть выполнен путем использования специального набора макрокоманд, предоставляемого операционной системой ДОС ЕС.

АСЕМБЛЕР В ДОС ЕС

6.1. ВЫПОЛНЕНИЕ ТРАНСЛЯТОРА АСЕМБЛЕРА

Для выполнения транслятора Ассемблера необходимо следующее оборудование:

операционная система ДОС ЕС;

14К байт основной памяти для транслятора Ассемблера Е и 44К байт для транслятора Ассемблера F;

набор внешних устройств (приведен в табл. 17).

Таблица 17

Имя логического устройства	Назначение устройства. Характеристика записей	Тип физического устройства	Имя файла
SYSRES SYSIPT	Резиденция системы Ввод исходной программы. Записи длиной 80 байт, неблокированные, фиксированной длины	Диск Устройство ввода перфокарт, магнитная лента, диск	IJSYSRS IJSYSIN
SYSSLB	Личная библиотека исходных модулей	Диск	IJSYSSL
SYSPCH	Вывод объектного модуля. Записи неблокированные, фиксированной длины, длиной 81 байт. Первый байт — управляющий символ выбора кармана, равный W. Устройство требуется, если задан режим DECK	Устройство вывода перфокарт, магнитная лента, диск	IJSYSPH
SYSLST	Вывод на печать результатов трансляции. Записи неблокированные, фиксированной длины, длиной 121 байт. Первый байт — управляющий символ печати	Устройство печати, магнитная лента, диск	IJSYSLS
SYSLNK	Вывод объектного модуля для Редактора. Устройство требуется, если задан режим LINK или CATAL	Диск	IJSYSLN
SYSOO1 SYSOO2 SYSOO3 SYSLOG	Рабочие файлы Вывод сообщений о ходе трансляции	Магнитная лента, диск Пишущая машинка, устройство печати	IJSYSO1 IJSYSO2 IJSYSO3

Задание для выполнения транслятора Ассемблера состоит из управляющих карт и исходных данных. Исходными данными для

него служит исходная программа, написанная на языке Ассемблера. Операторы, которые могут присутствовать в задании на трансляцию, приведены в табл. 18. Операторы указаны в том порядке, в каком они должны следовать в задании. Управляющие операторы вводятся с логического устройства SYSRDR, а исходная программа — с SYSIPT. Как правило, для них назначается одно и то же физическое устройство. Если этим устройством является диск, то оно должно называться SYSIN.

Не все операторы, перечисленные в табл. 18, должны обязательно присутствовать в задании на трансляцию. Как правило, в ДОС ЕС сгенерированы стандартные назначения для логических устройств, поэтому в задании должны быть только те операторы ASSGN, которые назначают для логических устройств физические устройства, отличающиеся от стандартных.

Таблица 18

Оператор	Назначение оператора
// JOB ИМЯ	Указывает идентификатор задания. Требуется всегда
// ASSGN SYSSLB,...	Назначает логическому устройству SYSSLB конкретное физическое устройство. Требуется в тех случаях, когда используется личная библиотека исходных модулей
// DLBL IJSYSSL,...	Задают информацию о метках и участках SYSSLB на диске. Требуется, если используется личная библиотека исходных модулей и отсутствует информация о ее метках и участках на цилиндре меток
// EXTENT SYSSLB,...	Назначает физическое устройство для SYSIPT
// ASSGN SYSIPT,...	Назначает физическое устройство для SYSIPT
// ASSGN SYSLST,...	Назначает физическое устройство для SYSLST
// ASSGN SYSPCH,...	Назначает физическое устройство для SYSPCH. Требуется, если задан режим DECK
// ASSGN SYSLNK,...	Назначает физическое устройство для SYSLNK. Требуется, если задан режим LINK или CATAL
// ASSGN SYS001,...	Назначают физическое устройство для рабочих файлов
// ASSGN SYS002,...	
// ASSGN SYS003,...	
// OPTION	Задаёт режимы работы транслятора Ассемблера
// EXEC ASSEMBLY	Вызывает для выполнения транслятор Ассемблера
* Карты исходной программы	Исходные данные для транслятора
/*	Указывает конец исходных данных
/&	Указывает конец задания

Устройства SYSSLB, SYSPCH и SYSLNK следует назначать только в том случае, если они используются при трансляции. Оператор OPTION необходим, если нужно изменить стандартные ре-

жимы. Логические устройства SYSIPT, SYSLST и SYSPCH могут назначаться на диск. При этом операторы DLBL и EXTENT должны указывать метки и участки на диске, отводимые для этих устройств. Устройство SYSLNK всегда, а рабочие файлы SYS001, SYS002 и SYS003 почти всегда, назначаются на диски. Если информация о метках и участках на дисках для этих файлов не присутствует на цилиндре меток, то ее нужно описать операторами DLBL и EXTENT.

Ниже приведен вариант задания для выполнения транслятора Ассемблера.

```
// JOB TRANS
// OPTION DECK
// ASSGN SYS001,X'192'
// ASSGN SYS002,X'192'
// ASSGN SYS003,X'192'
// DLBL IJSYS01 описание файла SYS001
// EXTENT SYS001,111111,8,1,100,500,4
// DLBL IJSYS02 описание файла SYS002
// EXTENT SYS002,111111,8,1,105,500,9
// DLBL IJSYS03 описание файла SYS003
// EXTENT SYS003,111111,,1,1100,500
// EXEC ASSEMBLY
      карты исходной программы
/*
/&
```

Предполагается, что в системе сгенерированы стандартные назначения для SYSIPT, SYSLST и SYSPCH, поэтому в задании не используются карты ASSGN.

Рабочие файлы SYS001, SYS002 и SYS003 назначены на диск, который нужно ставить на накопитель X'192'. Этот диск должен быть проинициализирован именем 111111. Для SYS001 и SYS002 отведено по 50 разделенных цилиндров. Начинаются эти файлы с цилиндра 10. Для SYS003 отведено 50 цилиндров, начиная с цилиндра 110.

Обычно в задании не нужно назначать физические устройства для рабочих файлов SYS001, SYS002 и SYS003. Давать информацию о метках и участках на диске для этих устройств также не следует, потому что эта информация, как правило, записана на цилиндре меток.

Логическое устройство SYSIPT можно назначить на магнитную ленту или диск. В этом случае исходные данные для транслятора записывают с помощью программ перезаписи соответственно на магнитную ленту или на диск. Устройства SYSPCH и SYSLST также можно назначать на магнитную ленту или диск. При этом содержимое магнитной ленты или диска после трансляции распечатывается или выводится на карты также с помощью программ перезаписи.

Все программы, которые входят в ДОС ЕС или выполняются под ее управлением, должны объединяться в библиотеке. Существуют три вида библиотек:

библиотека абсолютных модулей (CL);

библиотека объектных модулей (RL);

библиотека исходных модулей (SL).

Библиотеки, постоянно хранящиеся в резиденции системы, называются системными. К ним обращаются, не назначая физического устройства. Для хранения программ можно использовать также личные библиотеки объектных и исходных модулей, которые не входят в резиденцию системы. Их структура аналогична системным. При использовании личной библиотеки объектных модулей нужно назначать физическое устройство для логического устройства SYSRLB, а при использовании личной библиотеки исходных модулей — для SYSSLB.

В библиотеке абсолютных модулей находятся программы (фазы), готовые к выполнению.

Библиотеки объектных модулей содержат объектные модули — результат трансляции исходных модулей. Объектный модуль, который создается при работе транслятора Ассемблера, сразу после трансляции можно поместить (закаталогизировать) в библиотеку объектных модулей.

В библиотеке исходных модулей находятся исходные модули — книги, которые представляют собой последовательность операторов на любом языке программирования. Библиотека исходных модулей разделяется на подбиблиотеки. Подбиблиотека — это набор книг на исходном языке, предназначенных для использования определенным транслятором. Каждая подбиблиотека обозначается одной латинской буквой. Например подбиблиотека Ассемблера обозначается буквой А. В эту подбиблиотеку входят макроопределения и копируемые тексты (включаемые в программу по оператору COPY).

Для каталогизации макроопределений и копируемого текста используется программа Библиотекаря MAINT. Каталогизируемая книга должна начинаться с оператора CATALS. Его формат:

○○○CATALS○ подбиблиотека.имя книги

Для макроопределения имя книги должно совпадать с кодом операции прототипа, а для копируемого текста имя книги должно совпадать с тем именем, которое указывается в операторе COPY при вызове книги.

За оператором CATALS должно следовать макроопределение или копируемый текст, заключенный в операторы BKEND. Они записываются в таком формате:

○○○○○BKEND○○○

В задании, приведенном ниже, в системную библиотеку исходных модулей каталогизируется макроопределение M1.

```
// JOB CATMACRO
// EXEC MAINT
UUUUCATALS A.M1
    макроопределение
/•
/&
```

В следующем задании копируемый текст COPT1 помещается в библиотеку исходных модулей.

```
// JOB CATCOPT
// EXEC MAINT
UUUUCATALS A.COPT1
UUUUBKEND
    копируемый текст
UUUUBKEND
/•
/&
```

ПРИЛОЖЕНИЕ 1
СРАВНЕНИЕ АССЕМБЛЕРОВ

В данном приложении сравниваются Ассемблер Е ДОС ЕС, Ассемблер F ДОС ЕС и Ассемблер ОС ЕС.

Возможность	ДОС ЕС Ассемб- лер Е	ДОС ЕС Ассемб- лер F	ОС ЕС
Минимальный объем основной памяти, необходи- мый для выполнения транслятора	14К	44К	44К
Максимальное количество строк продолжения оператора (исключая макрокоманды и операто- ры прототипа)	1	2	2
Максимальное количество операндов в поле опе- рандов операторов DC и DS	1	32	32
Использование модификатора длины в битах Использование констант типа L (константа с пла- вающей точкой длиной до 16 байт)	Нет	Да	Да
Использование константы типа Q (адресная кон- станта, определяющая смещение внешней фик- тивной области)	Нет	Нет	Да
Использование оператора DXD ¹ (ОПРЕДЕЛИТЬ ВНЕШНЮЮ ФИКТИВНУЮ ОБЛАСТЬ)	Нет	Нет	Да
Использование оператора CXD ¹ (ОПРЕДЕЛИТЬ ОБЩУЮ ДЛИНУ ВНЕШНИХ ФИКТИВНЫХ ОБЛАСТЕЙ)	Нет	Нет	Да
Использование оператора WXTRN ² (ОПРЕДЕ- ЛИТЬ СЛАБОЕ ВНЕШНЕЕ ИМЯ)	Нет	Нет	Да
Использование оператора OPSYN ³ (ОПРЕДЕ- ЛИТЬ КОД ОПЕРАЦИИ)	Нет	Нет	Да
Максимальное количество операндов в макро- команде и операторе прототипа	100	200	200
Максимальное количество знаков в операнде макрокоманды	127	255	255
Максимальная длина значения знакового выра- жения	127	255	255
Стандартное количество переходов по операторам AGO и AIF	150	4096	4096

¹ Операторы DXD и CXD и константа типа Q позволяют использовать в разных исходных модулях внешние фиктивные области и указывать длину области памяти, отводимой для внешних фиктивных областей при выполнении программы. Внешние фиктивные области позволяют программисту определять рабочие области памяти в различных исходных модулях, а затем объединять их в один блок памяти, доступный каждому модулю.

² Оператор WXTRN Ассемблером обрабатывается так, как и оператор EXTRN.

ПРИЛОЖЕНИЕ 2

ОПЕРАТОРЫ ЯЗЫКА АССЕМБЛЕРА

В табл. 1 приведены мнемонические коды операций на языке Ассемблера всех машинных команд ЕС ЭВМ. Там же приведены некоторые сведения о возможностях записи машинных команд на языке Ассемблера. Графа «Команда» содержит название команды, графа «Код операции» содержит мнемонический и машинный коды операции машинной команды, графа «Формат команды» указывает формат машинной команды, графа «Формат операндов» показывает формат поля операндов машинной команды с использованием явных и неявных адресов, явной и неявной длины.

В табл. 1 используются следующие обозначения:

- R1, R2, R3 — абсолютные выражения, определяющие номера общих регистров 0—15 или номера регистров с плавающей точкой 0, 2, 4 и 6;
- X2 — абсолютное выражение, определяющее номер регистра индекса 0—15;
- B1, B2 — абсолютные выражения, определяющие номера регистров базы 0—15;
- D1, D2 — абсолютные выражения, определяющие смещение, которое должно находиться в пределах от 0 до 4095;
- L, L1, L2, — абсолютные выражения, определяющие длины операндов. Значение L должно быть в пределах от 0 до 256, значения L1 и L2 — в пределах от 0 до 16;
- I2 — абсолютное выражение, которое определяет непосредственный операнд машинной команды. Значение выражения должно быть в пределах от 0 до 255;
- S1, S2 — абсолютные и переместимые выражения, определяющие неявный адрес.

В табл. 2 приведены мнемонические коды и названия команд Ассемблера.

В табл. 3 приведены мнемонические коды и названия команд генерации.

В табл. 4 приведены сведения о том, как могут быть заполнены поля оператора при записи различных операторов Ассемблера.

Однако в зависимости от установленного режима редактирования Редактором элементы EXTRN и WXTN могут обрабатываться по-разному.

³ Оператор OPSYN позволяет на время трансляции исходного модуля удалить мнемонический код машинной команды или определить новый мнемонический код операции, по своим свойствам совпадающий с некоторым уже существующим мнемоническим кодом машинной команды.

Команда	Код операции		формат команды	формат операндов	
	мнемонический	машинный		явный адрес	неявный адрес
СЛОЖЕНИЕ	AR ^v	1A	RR	R1,R2	R1,S2(X2) или R1,S2
СЛОЖЕНИЕ	A	5A	RX	R1,D2(X2,B2), или R1,D2(B2)	R1,S2(X2) или R1,S2
СЛОЖЕНИЕ	AN ^v	4A	RX	R1,D2(X2,B2), или R1,D2(B2)	S1(L),S2(L2) или S1,S2
СЛОЖЕНИЕ	AP ^v	FA	SS	D1(L1,B1),D2(L2,B2) или D1(B1),D2(B2)	
СЛОЖЕНИЕ КОДОВ	ALR	1E	RR	R1,R2	
СЛОЖЕНИЕ КОДОВ	AL	5E	RX	R1,D2(X2,B2), или R1,D2(B2)	R1,S2(X2) или R1,S2
СЛОЖЕНИЕ С НОРМАЛИЗАЦИЕЙ (ДЛИННОЕ)	ADR	2A	RR	R1,R2	
СЛОЖЕНИЕ С НОРМАЛИЗАЦИЕЙ (ДЛИННОЕ)	AD	6A	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
СЛОЖЕНИЕ С НОРМАЛИЗАЦИЕЙ (КОРОТКОЕ)	AER	3A	RR	R1,R2	
СЛОЖЕНИЕ С НОРМАЛИЗАЦИЕЙ (КОРОТКОЕ)	AE	7A	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
СЛОЖЕНИЕ БЕЗ НОРМАЛИЗАЦИИ (ДЛИННОЕ)	AWR	2E	RR	R1,R2	
СЛОЖЕНИЕ БЕЗ НОРМАЛИЗАЦИИ (ДЛИННОЕ)	AW	6E	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
СЛОЖЕНИЕ БЕЗ НОРМАЛИЗАЦИИ (КОРОТКОЕ)	AUR	3E	RR	R1,R2	
СЛОЖЕНИЕ БЕЗ НОРМАЛИЗАЦИИ (КОРОТКОЕ)	AU	7E	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
И	NR	14	RR	R1,R2	
И	N	54	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
И	NI ^v	94	SI	D1(B1),D2(B2) или D1(L,B1),D2(B2)	S1,L2
И	NC ^v	D4	SS		S1(L),S2 или S1,S2
ПЕРЕХОД С ВОЗВРАТОМ	BALR ^v	05	RR	R1,R2	
ПЕРЕХОД С ВОЗВРАТОМ	BAL ^v	45	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
УСЛОВНЫЙ ПЕРЕХОД	BCR ^v	07	RR	R1,R2	

Команда	Код операции		Формат команды	Формат операндов	
	мнемонический	машинный		явный адрес	неявный адрес
УСЛОВНЫЙ ПЕРЕХОД	BC ✓	47	RX	R1,D2(X2,B2)	R1,S2(X2) или R1,S2
ПЕРЕХОД ПО СЧЕТЧИКУ	BSTR ✓	06	RR	R1,R2	
ПЕРЕХОД ПО СЧЕТЧИКУ	BST ✓	46	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
ПЕРЕХОД ПО ИНДЕКСУ БОЛЬШЕ	BXH	86	RS	R1,R3,D2(B2)	R1,R3,S2
ПЕРЕХОД ПО ИНДЕКСУ МЕНЬШЕ ИЛИ РАВНО	BXLE	87	RS	R1,R3,D2(B2)	R1,R3,S2
ПЕРЕХОД ПО «РАВНО»	BE ✓ ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД ПО «БОЛЬШЕ»	BE ✓ ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД ПО «МЕНЬШЕ»	BL ✓ ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД ПО «НЕ РАВНО»	BNE ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД ПО «НЕ БОЛЬШЕ»	BNI ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД ПО «НЕ МЕНЬШЕ»	BNI ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД ПО ЗНАКУ «-»	BM ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД ПО ЗНАКУ «+»	BP ✓ ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД ПО «НУЛЮ»	BZ ✓ ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД ПО «НЕ +»	BNM ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД ПО «НЕ 0»	BNZ ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД ПО ПЕРЕПОЛНЕНИЮ	BO ✓ ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД, ЕСЛИ ВСЕ 1	BO ✓ ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД, ЕСЛИ НЕ ВСЕ 1	BNO ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД, ЕСЛИ ЕСТЬ 1 И 0	BM ✓ ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
ПЕРЕХОД, ЕСЛИ ВСЕ 0	BZ ✓ ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
БЕЗУСЛОВНЫЙ ПЕРЕХОД	BR ✓	07	RR	R2	S2(X2) или S2
БЕЗУСЛОВНЫЙ ПЕРЕХОД	BI ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
НЕТ ОПЕРАЦИИ	NOP ✓	07	RR	R2	S2(X2) или S2
НЕТ ОПЕРАЦИИ	NOP ✓	47	RX	D2(X2,B2) или D2(B2)	S2(X2) или S2
СРАВНЕНИЕ	CR ✓	19	RR	R1,R2	
СРАВНЕНИЕ	C	59	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2

Команда	Код операции		Формат команды	Формат операндов	
	микропроцессный	машинный		явный адрес	неявный адрес
СРАВНЕНИЕ ПОЛУСЛОВА СРАВНЕНИЕ ДЕСЯТИЧНОЕ	CH✓ CP✓	49 F9	RX SS	R1,D2(X2,B2) или R1,D2(B2) D1(L1,B1),D2(L2,B2) или D1(B1),D2(B2)	R1,S2(X2) или R1,S2 S1(L1),S2(L2) или S1,S2
	CLR CL CL1✓ CLC✓	15 55 95 D5	RR RX SI SS	R1,R2 R1,D2(X2,B2) или R1,D2(B2) D1(B1),D2(B2) или D1(L1,B1),D2(B2)	R1,S2(X2) или R1,S2 S1,12 S1(L),S2 или S1,S2
СРАВНЕНИЕ КОДОВ СРАВНЕНИЕ КОДОВ СРАВНЕНИЕ КОДОВ	CDR CD CER CE	29 69 39 79	RR RX RR RX	R1,R2 R1,D2(X2,B2) или R1,D2(B2) R1,R2 R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2 R1,S2(X2) или R1,S2 R1,S2(X2) или R1,S2
	CVB✓ CVD✓	4F 4E	RX RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
ПРЕОБРАЗОВАНИЕ В ДВОИЧ- НУЮ ПРЕОБРАЗОВАНИЕ В ДЕСЯТИЧ- НУЮ	DR D DP✓	1D 5D FD	RR RX SS	R1,R2 R1,D2(X2,B2) или R1,D2(B2) D1(L1,B1),D2(L2,B2) или D1(B1),D2(B2)	R1,S2(X2) или R1,S2 S1(L1),S2(L2) или S1,S2
	DDR DD DER DE ED✓	2D 6D 3D 7D DE	RR RX RR RX SS	R1,R2 R1,D2(X2,B2) или R1,D2(B2) R1,R2 R1,D2(X2,B2) или R1,D2(B2) D1(L1,B1),D2(B2) или D1(B1),D2(B2)	R1,S2(X2) или R1,S2 R1,S2(X2) или R1,S2 S1(L),S2 или S1,S2
ОТРЕДАКТИРОВАТЬ И ОТМЕТИТЬ ОТРЕДАКТИРОВАТЬ ИЛИ ОТМЕТИТЬ	EDMK EX XR X	DF 44 17 57	SS RX RR RX	или D1(B1),D2(B) R1,D2(X2,B2) или R1,D2(B2) R1,R2 R1,D2(X2,B2) или R1,D2(B2)	S1(L),S2 или S1,S2 R1,S2(X2) или R1,S2 R1,S2(X2) или R1,S2 R1,S2(X2) или R1,S2

Команда	Код операции		Формат команды	Формат операндов	Формат адрес	Формат адрес
	мнемонический	машинный				
ИСКЛЮЧАЮЩЕЕ ИЛИ ИСКЛЮЧАЮЩЕЕ ИЛИ	XI√ XC√	97 D7	SI SS	D1(B1),12 D1(L,B1),D2(B2) или D1(B1),D2(B2)	SI,12 SI(L),S2 или SI,S2	
ПОПОЛАМ (ДЛИННОЕ) ПОПОЛАМ (КОРОТКОЕ) ОСТАНОВИТЬ ВВОД — ВЫВОД ПРОЧИТАТЬ СИМВОЛ ПРОЧИТАТЬ КЛЮЧ ПАМЯТИ	HDR HER HIO IC	24 34 9E 43	RR RR SI RX	R1,R2 R1,R2 D1(B1) R1,D2(X2,B2) или R1,R2	SI R1,S2(X2) или R1,S2	
ЗАГРУЗКА ЗАГРУЗКА АДРЕСА ЗАГРУЗКА ПОЛУСЛОВА ЗАГРУЗКА (ДЛИННАЯ) ЗАГРУЗКА (ДЛИННАЯ) ЗАГРУЗКА (КОРОТКАЯ) ЗАГРУЗКА (КОРОТКАЯ) ЗАГРУЗКА ГРУППОВАЯ ЗАГРУЗКА И ПРОВЕРКА ЗАГРУЗКА И ПРОВЕРКА (ДЛИН- НАЯ)	ISK LRV L LA√ LHV LDR LD LER LE LMV LTR LTDR	09 18 58 41 48 28 68 38 78 98 12 22	RR RX RX RX RX RX RX RX RX RS RR RR	R1,R2 R1,D2(X2,B2) R1,D2(X2,B2) R1,D2(X2,B2) или R1,R2 R1,D2(X2,B2) R1,D2(X2,B2) или R1,R3,D2(B2) R1,R2 R1,R2	R1,S2(X2) или R1,S2 R1,S2(X2) или R1,S2 R1,S2(X2) или R1,S2 R1,S2(X2) или R1,S2 R1,S2(X2) или R1,S2 R1,R3,S2	
ЗАГРУЗКА И ПРОВЕРКА (КОРОТ- КАЯ) ЗАГРУЗКА, ДОПОЛНЕНИЯ ЗАГРУЗКА ДОПОЛНЕНИЯ (ДЛИННАЯ) ЗАГРУЗКА ДОПОЛНЕНИЯ (КО- РОТКАЯ) ЗАГРУЗКА ПОЛОЖИТЕЛЬНАЯ ЗАГРУЗКА ПОЛОЖИТЕЛЬНАЯ (ДЛИННАЯ)	LTER LCR LCDR LCER LPR LPDR	32 13 23 33 10 20	RR RR RR RR RR RR	R1,R2 R1,R2 R1,R2 R1,R2 R1,R2 R1,R2		

Команда	Код операции		Формат команды	Формат операндов	
	мнемонический	машинный		явный адрес	неявный адрес
ЗАГРУЗКА ПОЛОЖИТЕЛЬНАЯ (КОРОТКАЯ)	LPER	30	RR	R1,R2	
ЗАГРУЗКА ОТРИЦАТЕЛЬНАЯ	LNR	11	RR	R1,R2	
ЗАГРУЗКА ОТРИЦАТЕЛЬНАЯ (ДЛИННАЯ)	LNDR	21	RR	R1,R2	
ЗАГРУЗКА ОТРИЦАТЕЛЬНАЯ (КОРОТКАЯ)	LNER	31	RR	R1,R2	
ЗАГРУЗКА PSW	LPSW	82	SI	D1(B1)	SI
ПЕРЕСЫЛКА СИМВОЛА	MVIV	92	SI	D1(B1),12	SI,12
ПЕРЕСЫЛКА	MVCV	D2	SS	D1(L,B1),D2(B2) или D1(B1),D2(B2)	SI(L),S2 или SI,S2
ПЕРЕСЫЛКА ЦИФ	MVN	D1	SS	D1(L,B1),D2(B2) или D1(B1),D2(B2)	SI(L),S2 или SI,S2
ПЕРЕСЫЛКА ЗОН	MVZV	D3	SS	D1(L,B1),D2(B2) или D1(B1),D2(B2)	SI(L),S2 или SI,S2
ПЕРЕСЫЛКА СО СДВИГОМ	MVO	F1	SS	D1(L1,B1),D2(L2,B2) или D1(B1),D2(B2)	SI(L1),S2(L2) или SI,S2
УМНОЖЕНИЕ	MR	1C	RR	R1,R2	
УМНОЖЕНИЕ	M	5C	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
УМНОЖЕНИЕ ПОЛУСЛОВА	MN	4C	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
УМНОЖЕНИЕ ДЕСЯТИЧНОЕ	MP	FC	SS	D1(L1,B1),D2(L2,B2) или D1(B1),D2(B2)	SI(L1),S2(L2) или SI,S2
УМНОЖЕНИЕ (ДЛИННОЕ)	MDR	2C	RR	R1,R2	
УМНОЖЕНИЕ (ДЛИННОЕ)	MD	6C	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
УМНОЖЕНИЕ (КОРОТКОЕ)	MER	3C	RR	R1,R2	
УМНОЖЕНИЕ (КОРОТКОЕ)	ME	7C	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
ИЛИ	OR	16	RR	R1,R2	
ИЛИ	O	56	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
ИЛИ	OIV	96	SI	D1(B1),12	SI,12
ИЛИ	OCV	D6	SS	D1(L,B1),D2(B2) или D1(B1),D2(B2)	SI(L),S2 или SI,S2

Команда	Код операции		Формат команды	Формат операторов	
	мнемонический	машинный		явный адрес	неявный адрес
УПАКОВАТЬ	PACKV	F2	SS	D1(L1,B1),D2(L2,B2) или D1(B1),D2(B2)	S1(L1),S2(L2) или S1,S2
ПРЯМОЕ ЧТЕНИЕ	RDD	85	SI	D1(B1),12	S1,12
УСТАНОВИТЬ МАСКУ ПРОГРАММЫ	SPMV	04	RR	R1	SI
УСТАНОВИТЬ МАСКУ СИСТЕМЫ	SSM	80	SI	D1(B1)	SI
УСТАНОВИТЬ КЛЮЧ ПАМЯТИ	SSK	08	RR	R1,R2	SI
НАЧАТЬ ВВОД — ВЫВОД	SIOV	9C	SI	D1(B1)	SI
СДВИГ ВЛЕВО	SLA	8B	RS	R1,D2(B2)	R1,S2
СДВИГ ВЛЕВО ДВОЙНОЙ	SLDA	8F	RS	R1,D2(B2)	R1,S2
СДВИГ ВЛЕВО КОДОВ	SLLV	89	RS	R1,D2(B2)	R1,S2
СДВИГ ВЛЕВО ДВОЙНОЙ КОДОВ	SLDL	8D	RS	R1,D2(B2)	R1,S2
СДВИГ ВПРАВО	SRA	8A	RS	R1,D2(B2)	R1,S2
СДВИГ ВПРАВО ДВОЙНОЙ	SRDA	8E	RS	R1,D2(B2)	R1,S2
СДВИГ ВПРАВО КОДОВ	SRLV	88	RS	R1,D2(B2)	R1,S2
СДВИГ ВПРАВО ДВОЙНОЙ КОДОВ	SRDL	8C	RS	R1,D2(B2)	R1,S2
ЗАПИСЬ В ПАМЯТЬ	ST	50	RX	R1,D2(X2,B2)	R1,S2(X2) или R1,S2
ЗАПИСЬ В ПАМЯТЬ ПОЛУСЛОВА	STHV	40	RX	R1,D2(X2,B2)	R1,S2(X2) или R1,S2
ЗАПИСЬ В ПАМЯТЬ СИМВОЛА	STC	42	RX	R1,D2(X2,B2)	R1,S2(X2) или R1,S2
ЗАПИСЬ В ПАМЯТЬ ГРУППОВАЯ	STMV	90	RS	R1,R3,D2(B2)	R1,R3,S2
ЗАПИСЬ В ПАМЯТЬ (ДЛИННАЯ)	STD	60	RX	R1,D2(X2,B2)	R1,S2(X2) или R1,S2
ЗАПИСЬ В ПАМЯТЬ (КОРОТКАЯ)	STE	70	RX	R1,D2(X2,B2)	R1,S2(X2) или R1,S2
ВЫЧИТАНИЕ	SRV	1B	RR	R1,R2	R1,S2(X2) или R1,S2
ВЫЧИТАНИЕ ПОЛУСЛОВА	SHV	5B	RX	R1,D2(X2,B2)	R1,S2(X2) или R1,S2
ВЫЧИТАНИЕ ДЕСЯТИЧНОЕ	SPV	4B	RX	R1,D2(X2,B2)	R1,S2(X2) или R1,S2
		FB	SS	D1(L1,B1),D2(L2,B2) или D1(B1),D2(B2)	S1(L1),S2(L2) или S1,S2
ВЫЧИТАНИЕ КОДОВ	SLR	1F	RR	R1,R2	R1,S2(X2) или R1,S2
ВЫЧИТАНИЕ КОДОВ	SL	5F	RX	R1,D2(X2,B2)	R1,S2(X2) или R1,S2

Команда	Код операции		Формат команд	Формат operands	
	мнемонический	машинный		язык адрес	язык адрес
ВЫЧИТАНИЕ С НОРМАЛИЗАЦИЕЙ (ДЛИННОЕ)	SDR	2B	RR	R1,R2	R1,R2
ВЫЧИТАНИЕ С НОРМАЛИЗАЦИЕЙ (ДЛИННОЕ)	SD	6B	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
ВЫЧИТАНИЕ С НОРМАЛИЗАЦИЕЙ (КОРОТКОЕ)	SER	3B	RR	R1,R2	
ВЫЧИТАНИЕ С НОРМАЛИЗАЦИЕЙ (КОРОТКОЕ)	SE	7B	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
ВЫЧИТАНИЕ БЕЗ НОРМАЛИЗАЦИИ (ДЛИННОЕ)	SWR	2F	RR	R1,R2	
ВЫЧИТАНИЕ БЕЗ НОРМАЛИЗАЦИИ (ДЛИННОЕ)	SW	6F	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
ВЫЧИТАНИЕ БЕЗ НОРМАЛИЗАЦИИ (КОРОТКОЕ)	SUR	3F	RR	R1,R2	
ВЫЧИТАНИЕ БЕЗ НОРМАЛИЗАЦИИ (КОРОТКОЕ)	SU	7F	RX	R1,D2(X2,B2) или R1,D2(B2)	R1,S2(X2) или R1,S2
ОБРАЩЕНИЕ К СУПЕРВИЗОРУ	SVC ✓	OA	RR	I2	
ОПРОСИТЬ КАНАЛ	TCH	9F	SI	D1(B1)	S1
ПРОПИСЬ ВВОД-ВЫВОД	TIO ✓	9D	SI	D1(B1)	S1
ПРОВЕРИТЬ И УСТАНОВИТЬ	TS	93	SI	D1(B1)	S1
ПРОВЕРИТЬ ПО МАСКЕ	TM ✓	91	SI	D1(B1),I2	S1,I2
ПЕРЕКОДИРОВАТЬ	TR ✓	DC	SS	D1(L,B1),D2(B2) или D1(B1),D2(B2)	S1(L),S2 или S1,S2
ПЕРЕКОДИРОВАТЬ И ПРОВЕРИТЬ	TRT ✓	DD	SS	D1(L,B1),D2(B2) или D1(B1),D2(B2)	S1(L),S2 или S1,S2
РАСПАКОВАТЬ	UNPK ✓	F3	SS	D1(L,B1),D2(L2,B2) или D1(B1),D2(B2)	S1(L1),S2(L2) или S1,S2
ПРЯМАЯ ЗАПИСЬ	WRD	84	SI	D1(B1),I2	S1,I2
СЛОЖЕНИЕ С ОЧИСТКОЙ	ZAP ✓	F8	SS	D1(L,B1),D2(L2,B2) или D1(B1),D2(B2)	S1(L1),S2(L2) или S1,S2

* Расширенный мнемонический код.

Таблица 2

Команда Ассемблера	Мнемонический код
ОПРЕДЕЛИТЬ КОМАНДУ ВВОДА—ВЫВОДА	CCW
УСТАНОВИТЬ ГРАНИЦУ	CNOP
ОПРЕДЕЛИТЬ ОБЩУЮ ОБЛАСТЬ	COM ✓
КОПИРОВАТЬ КНИГУ	COPY ✓
ОПРЕДЕЛИТЬ ПРОГРАММНУЮ СЕКЦИЮ	CSECT ✓
ОПРЕДЕЛИТЬ КОНСТАНТУ	DC ✓
ОТМЕНИТЬ РЕГИСТР БАЗЫ	DROP ✓
ОПРЕДЕЛИТЬ ПАМЯТЬ	DS ✓
ОПРЕДЕЛИТЬ ФИКТИВНУЮ ОБЛАСТЬ	DSECT ✓
НАЧАТЬ НОВУЮ СТРАНИЦУ	EJECT ✓
ЗАКОНЧИТЬ МОДУЛЬ	END ✓
ОПРЕДЕЛИТЬ ВХОДНОЕ ИМЯ	ENTRY ✓
ПРИСВОИТЬ ЗНАЧЕНИЕ	EQU ✓
ОПРЕДЕЛИТЬ ВНЕШНЕЕ ИМЯ	EXTRN ✓
УПРАВЛЯТЬ ФОРМАТОМ ВВОДА	ICTL ✓
ПРОВЕРИТЬ НУМЕРАЦИЮ КАРТ	ISEQ ✓
НАЧАТЬ ОБЛАСТЬ ЛИТЕРАЛОВ	LTORG ✓
УСТАНОВИТЬ СЧЕТЧИК АДРЕСА	ORG ✓
УПРАВЛЯТЬ ПЕЧАТЬЮ	PRINT ✓
ПЕРФОРИРОВАТЬ КАРТУ	PUNCH ✓
ПЕРФОРИРОВАТЬ СЛЕДУЮЩУЮ КАРТУ	REPRO ✓
ПРОПУСТИТЬ СТРОКУ	SPACE ✓
УСТАНОВИТЬ НАЧАЛО ПРОГРАММЫ	START ✓
ИДЕНТИФИЦИРОВАТЬ ВЫВОД	TITLE ✓
ОПРЕДЕЛИТЬ РЕГИСТР БАЗЫ	USING ✓

Таблица 3

Команда генерации	Мнемонический код
УСТАНОВИТЬ ЧИСЛО ПЕРЕХОДОВ	ACTR ✓
БЕЗУСЛОВНЫЙ ПЕРЕХОД	AGO ✓
УСЛОВНЫЙ ПЕРЕХОД	AIF ✓
ОПРЕДЕЛИТЬ ИМЯ ПЕРЕХОДА	ANOP ✓
ОПРЕДЕЛИТЬ ГЛОБАЛЬНЫЙ АРИФМЕТИЧЕСКИЙ ПЕРЕМЕННЫЙ ПАРАМЕТР	GBLA ✓
ОПРЕДЕЛИТЬ ГЛОБАЛЬНЫЙ ЛОГИЧЕСКИЙ ПЕРЕМЕННЫЙ ПАРАМЕТР	GBLB ✓
ОПРЕДЕЛИТЬ ГЛОБАЛЬНЫЙ ЗНАКОВЫЙ ПЕРЕМЕННЫЙ ПАРАМЕТР	GBLC ✓
ОПРЕДЕЛИТЬ ЛОКАЛЬНЫЙ АРИФМЕТИЧЕСКИЙ ПЕРЕМЕННЫЙ ПАРАМЕТР	LCLA ✓
ОПРЕДЕЛИТЬ ЛОКАЛЬНЫЙ ЛОГИЧЕСКИЙ ПЕРЕМЕННЫЙ ПАРАМЕТР	LCLB ✓
ОПРЕДЕЛИТЬ ЛОКАЛЬНЫЙ ЗНАКОВЫЙ ПЕРЕМЕННЫЙ ПАРАМЕТР	LCLC ✓
НАЧАЛО МАКРООПРЕДЕЛЕНИЯ	MACRO ✓
КОНЕЦ МАКРООПРЕДЕЛЕНИЯ	MEND ✓
ВЫЙТИ ИЗ МАКРООПРЕДЕЛЕНИЯ	MEXIT ✓

Команда генерации	Мнемонический код
СООБЩИТЬ ОБ ОШИБКЕ УСТАНОВИТЬ АРИФМЕТИЧЕСКОЕ ЗНАЧЕНИЕ УСТАНОВИТЬ ЛОГИЧЕСКОЕ ЗНАЧЕНИЕ УСТАНОВИТЬ ЗНАКОВОЕ ЗНАЧЕНИЕ	MNOTE✓ SETA✓ SETB✓ SETC✓

Таблица 4

Поле операции	Поле названия	Поле операндов
Машинная команда	Любое имя ¹ или пробел	Операнды, разделенные запятыми. Для записи операндов используются абсолютные или переместимые выражения ⁶
Параметр или соединение параметров с другими знаками ² Прототип ³	Любое имя ¹ или пробел	Любая комбинация знаков (включая параметры)
Макрокоманда	Постоянный параметр или пробел	Операнды, которые являются постоянными параметрами и (или) ключевыми параметрами, разделенные запятыми
ACTR AGO	Любое имя ⁴ или пробел	Позиционные операнды ⁴ или операнды ⁴ ключевого слова (вида: ключевое слово, знак равенства, значение), разделенные запятыми, или пробел
AIF	Имя перехода или пробел	Арифметическое выражение Имя перехода
ANOP CCW	Имя перехода Любое имя ¹ или пробел	Логическое выражение, заключенное в скобки, за которым следует имя перехода Не используется ⁵
CNOP	Имя перехода или пробел	Четыре операнда ⁶ , разделенные запятыми
COM	Имя перехода или пробел	Два абсолютных выражения ⁶ , разделенные запятой
COPY CSECT	Имя перехода Пробел Любое имя ¹ или пробел	Не используется ⁵ Простое имя Не используется ⁵
DC	Любое имя ¹ или пробел	От одного до 32 операндов ⁶ , разделенных запятыми
DROP	Имя перехода или пробел	От одного до 16 абсолютных выражений ⁶ , разделенных запятыми
DS	Любое имя ¹ или пробел	От одного до 32 операндов ⁶ , разделенных запятыми
DSECT	Параметр, простое имя или соединение параметра с другими знаками	Не используется ⁵

Поле операции	Поле названия	Поле операндов
EJECT	Имя перехода или пробел	Не используется ⁵
END	Имя перехода или пробел	Переместимое выражение ⁶ или пробел
ENTRY	Имя перехода или пробел	Одно или более переместимых имен ⁶ , разделенных запятыми
EQU	Параметр, простое имя или соединение параметров с другими знаками	Абсолютное или переместимое выражение ⁶
EXTRN	Имя перехода или пробел	Одно или более переместимых имен ⁶ , разделенных запятыми
GBLA, GBLB, GBLC	Пробел	Один или более переменных параметров ⁷ , разделенных запятыми
ICTL	Пробел	От одного до трех десятичных чисел, разделенных запятыми
ISEQ	Пробел	Два десятичных числа, разделенных запятыми
LTORG	Любое имя ¹ или пробел	Не используется ⁵
MACRO ³	Пробел	Не используется ⁵
MEND ³ , MEXIT ³	Имя перехода или пробел	Не используется ⁵
MNOTE ³	Имя перехода или пробел	Код ошибки, за которым следует запятая и любая комбинация знаков ⁶ , заключенная в апострофы
ORG	Имя перехода или пробел	Переместимое выражение ⁶ или пробел
PRINT	Имя перехода или пробел	От одного до трех операндов ⁶ , разделенных запятыми
PUNCH	Имя перехода или пробел	От одного до 80 знаков, заключенных в апострофы ⁶
REPRO ⁶	Имя перехода или пробел	Не используется ⁵
SETA	Арифметический параметр	Арифметическое выражение
SETB	Логический параметр	Логическое выражение
SETC	Знаковый параметр	Знаковое выражение
SPACE	Имя перехода или пробел	Десятичный самоопределенный терм ⁶ или пробел
START	Любое имя ¹ или пробел	Самоопределенный терм ⁶ или пробел
TITLE	Специальное имя (от 0 до 4 знаков), которое может генерироваться, имя перехода или пробел	От одного до 100 знаков, заключенных в апострофы ⁶

Поле операции	Поле названия	Поле операндов
USING	Имя перехода или пробел	Абсолютное или переместимое выражение, за которым следует от 1 до 16 абсолютных выражений ⁶

¹ Под любым именем подразумевается: простое имя, имя перехода, а также параметр или соединение параметров с другими знаками, которые эквивалентны простому имени.

² Не рекомендуется использовать параметры для генерации следующих кодов операций: COPY, END, ICTL, CSECT, DSECT, ISEQ, PRINT, REPRO, START, команд генерации и макрокоманд.

³ Оператор может использоваться только в макроопределении.

⁴ Перед обработкой макрокоманды все параметры, которые используются в поле названия или в поле операндов, заменяются значениями, им соответствующими.

⁵ Любая запись в поле операндов трактуется как комментарий.

⁶ Для генерации операндов могут использоваться параметры и соединение параметров с другими знаками.

⁷ Переменные параметры могут быть определены как индексированные переменные параметры.

⁸ В строке, следующей за оператором REPRO, не происходит замены параметров.

Адрес:

- базовый 34
- байта 9
- группы байт 9
- неявный 49
- явный 49

Адресация:

- литералов 163
- относительная 52

Алфавит языка Ассемблера 38

Апострофы парные 214

Базирование 56

Бланк кодирования 5

Выравнивание 140, 145

Выражение:

- абсолютное 29
- арифметическое 222
- знаковое 229
- логическое 236
- переместимое 29
 - простое 30
 - составное 31

Генерация 206

Граница группы байт 9

Данное 6

Длина:

- явная 50
- неявная 50

Значение:

- знаковое пустое 212
- символического имени 22
- счетчика адреса 24

Идентификация строки 45

Имя символическое:

- внешнее 199
- входное 199
- любое 20
- перехода 20
- простое 20

Индекс 70

Индикатор 131

Код операции мнемонический 15

— — — расширенный 32

Колонка:

- конца 38
- начала 38
- продолжения 38
- указателя продолжения 39

Команда:

- Ассемблера 13,33
- генерации 13
- десятичная 82
- логическая 97
- машинная 13,32
- определения 13,34
- перехода 117
- секционирования и соединения 35
- с плавающей точкой 92
- с фиксированной точкой 74
- управления 35

Комментарий 36

Константа:

- адресная 156
- внешняя 159
- двоичная 148
- десятичная 149
- знаковая 146
- с плавающей точкой 153
- с фиксированной точкой 151
- шестнадцатеричная 147

Кратность 143

— нулевая 167

Литерал 27, 160

— дублирующий 162

Макрокоманда:

- ключевая 212, 213
- позиционная 212
- смешанная 214

Макроопределение:

- ключевое 209
- позиционное 208
- смешанное 209

Макросредства 36

Мантисса числа 7, 92

Масштабирование 152, 154

Модификатор:

- длины 145

в битах 145, 155
 масштаба 145, 152
 порядка 145, 152
Модуль:
 исходный 12
 объектный 12
 абсолютный 276

Область:
 литералов 162
 общая 196
 фиктивная 191
Операнд:
 макрокоманды 212
 опущенный 212
 непосредственный 10
Оператор:
 ACTR 246
 AGO 245
 AIF 240
 ANOP 244
 COM 196
 COPY 180
 CNOP 140
 CSECT 184
 DC 143
 DROP 68
 DS 164
 DSECT 191
 EIECT 175
 END 182
 ENTRY 199
 EQU 170
 EXTRN 200
 GBLA 248
 GBLB 248
 GBLC 248
 ICTL 42
 ISEQ 46
 LCLA 221
 LCLB 221
 LCLC 221
 LTORG 162
 MACRO 207
 MEND 207
 MEXIT 255
 MNOTE 253
 ORG 138, 186
 прототипа 209
 PRINT 176
 PUNCH 179
 REPRO 179
 SETA 222
 SETB 236
 SETC 229
 SPACE 175
 START 137, 183
 TITLE 171
 USING 53

— в многосекционной програм-
 ме 187
Операция 6
**Определение символического име-
 ни 21**
 — предварительное 23
Отношение:
 арифметическое 237
 знаковое 238
Параметр:
 переменный 220
 — арифметический 222
 — глобальный 248
 — знаковый 229
 — индексированный 251
 — логический 236
 — локальный 221
 постоянный 208
 системный 216
 &SYSECT 217
 &SYSLIST 219
 &SYSNDX 216
Перемещение программы 17
Подстрока знаков 230
Поле:
 идентификации 45
 названия 13
 операндов 13
 операции 13
Порядок числа 7, 92
Признак переместимости 17
**Распечатка результатов трансля-
 ции 16**
Регистр базы 47
 — доступный 56
Редактирование данных 105
Секционирование 35
Секция:
 неименованная 187
 программная 183
Скобки парные 214
Словарь внешних имен 199
Слово ключевое 213
Смещение 47
Соединение программ 35
Список 215
Ссылка на характеристику длины 26
Строка:
 бланка кодирования 13, 38
 знаковая 229
 продолжения 38
Счетчик адреса 15, 24
 — в многосекционной програм-
 ме 184

Текст копируемый 180
Терм:
 абсолютный 17
 переместимый 17
 самоопределенный 17

- двоичный 17
- десятичный 17
- знаковый 17
- шестнадцатеричный 17

Термы спаренные 29

Тип константы 145

Формат:

- данного 7
- с зоной 8
- упакованный 8
- команды 9

Характеристика:

- длины 17
- выражения 29
- символического имени 22

— счетчика адреса 25

количества знаков 225

— операндов 226

типа 231

числа с плавающей точкой 8, 92

Цикл 125

Цифра:

- десятичная 5
- шестнадцатеричная 5

Часть программы основная 211

Число:

- десятичное 8
- с плавающей точкой 7
- с фиксированной точкой 7

ПРЕДИСЛОВИЕ	3
Глава 1. ОБЩИЕ СВЕДЕНИЯ О ДАННЫХ, КОМАНДАХ И АССЕМБ- ЛЕРЕ ЕС ЭВМ	5
1.1. Шестнадцатеричная система счисления	5
1.2. Форматы данных и команд в ЕС ЭВМ	6
1.3. Символическое программирование	11
1.4. Краткая характеристика Ассемблера	12
1.4.1. Назначение Ассемблера	12
1.4.2. Основные положения языка Ассемблера	13
1.4.3. Трансляция	15
Глава 2. ЭЛЕМЕНТЫ ЯЗЫКА АССЕМБЛЕРА	17
2.1. Термы и выражения	17
2.1.1. Самоопределенные термы	17
2.1.2. Символическое имя	20
2.1.3. Значение счетчика адреса	24
2.1.4. Ссылка на характеристику длины	26
2.1.5. Литералы	27
2.1.6. Составление и вычисление выражений	27
2.1.7. Типы выражений	29
2.2. Машинные команды	32
2.3. Команды Ассемблера	33
2.3.1. Команды определения	34
2.3.2. Команды секционирования и соединения	35
2.3.3. Команды управления	35
2.4. Макросредства	36
2.5. Комментарии	36
Глава 3. РЕКОМЕНДАЦИИ ПО ПРОГРАММИРОВАНИЮ	38
3.1. Запись операторов языка Ассемблера	38
3.1.1. Алфавит языка Ассемблера	38
3.1.2. Бланк кодирования	38
3.1.3. Правила записи операторов	39
3.1.4. Примеры записи операторов на бланке кодирования	41
3.1.5. Изменение границ операторов	42
3.1.6. Идентификация	45
3.2. Адресация памяти	47
3.2.1. Представление адреса в ЕС ЭВМ	47
3.2.2. Преимущества адресации с регистром базы	48
3.2.3. Способы адресации в Ассемблере	49
3.3. Использование операторов USING и DROP	53
3.3.1. Функции оператора USING	53
3.3.2. Программирование с оператором USING	55
3.3.3. Загрузка регистров базы	64
3.3.4. Функции оператора DROP	68
3.4. Использование машинных команд	69
3.4.1. Правила записи машинных команд	69
3.4.2. Команды с фиксированной точкой	74

3.4.3. Десятичные команды	82
3.4.4. Команды с плавающей точкой	92
3.4.5. Логические команды	97
3.4.6. Способы организации переходов в программе	117
3.4.7. Работа с подпрограммами	132
3.5. Работа со значением счетчика адреса	137
3.6. Определение и использование данных	142
3.6.1. Оператор определения констант	143
3.6.2. Примеры определения и использования констант	146
3.6.3. Адресные константы	156
3.7. Программирование с использованием литералов	160
3.8. Резервирование и описание областей памяти	164
3.8.1. Оператор определения области памяти	164
3.8.2. Нулевая кратность	167
3.9. Использование оператора EQU	170
3.10. Возможности воздействия на формат вывода результатов трансляции	171
3.10.1. Управление выводом распечатки	172
3.10.2. Воздействие на содержание распечатки	176
3.10.3. Вывод информации, дополнительной к объектному модулю	178
3.11. Секционирование и соединение программ	181
3.11.1. Деление на программные секции	183
3.11.2. Определение регистров базы в многосекционной программе	187
3.11.3. Использование фиктивных областей	191
3.11.4. Использование общих областей	196
3.11.5. Символическая связь между исходными модулями	198

Глава 4. МАКРОСРЕДСТВА 204

4.1. Возможности макросредств	204
4.2. Макроопределение	206
4.2.1. Состав макроопределения	207
4.2.2. Оператор прототипа	208
4.2.3. Соединение параметров с другими знаками	210
4.3. Макрокоманда	211
4.3.1. Операнды макрокоманды	212
4.3.2. Список операндов	215
4.4. Системные параметры	216
4.4.1. Параметр &SYSNDX	216
4.4.2. Параметр &SYSECT	217
4.4.3. Параметр &SVSLIST	219
4.5. Переменные параметры	220
4.5.1. Арифметические параметры	222
4.5.2. Знаковые параметры	229
4.5.3. Логические параметры	236
4.6. Генераторные переходы	239
4.6.1. Оператор AIF	240
4.6.2. Оператор AGO	245
4.6.3. Оператор ACTR	246
4.7. Дополнительные возможности макросредств	248
4.7.1. Глобальные параметры	248
4.7.2. Индексируемые переменные параметры	251
4.7.3. Оператор MNOTE	253
4.7.4. Оператор MEXIT	255
4.8. Примеры составления макроопределений	256
4.8.1. Извлечение корня	256
4.8.2. Генерация констант типа C	258
4.8.3. Пересылка содержимого области памяти	261
4.8.4. Генерация знаков констант в обратном порядке	263

Глава 5. ПРОГРАММИРОВАНИЕ ВВОДА — ВЫВОДА	265
5.1. Система ввода — вывода	265
5.1.1. Физический уровень управления вводом — выводом	266
5.1.2. Логический уровень управления вводом — выводом	266
5.1.3. Организация данных	267
5.2. Организация ввода — вывода на языке Ассемблера	267
Глава 6. АССЕМБЛЕР В ДОС ЕС	273
6.1. Выполнение транслятора Ассемблера	273
6.2. Использование библиотек ДОС ЕС	276
ПРИЛОЖЕНИЯ	278
Приложение 1. Сравнение Ассемблеров	278
Приложение 2. Операторы языка Ассемблера	279
Предметный указатель	291

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА ЕС ЭВМ

Редакторы Л. Д. Григорьева и Л. А. Табакова

Техн. редактор В. А. Чуракова. Корректор Г. А. Башарина

Худ. редактор Т. В. Стихно.

Переплет художника Ф. К. Мороз

Подписано к печати 28/XI 1975 г. Формат бумаги 60×90¹/₁₆. Бумага № 3.
 Объем 18,5 печ. л. Уч.-изд. л. 19,15. Тираж 40 000 экз. доп. А 03221.
 (Тематич. план 1975 г. № 72). Заказ № 2645. Цена 1 р. 17 к.

Издательство «Статистика», Москва, ул. Кирова, 39.

Отпечатано с матриц в областной типографии управления издательств,
 полиграфии и книжной торговли Ивановского облисполкома,
 г. Иваново-8, ул. Типографская, 6.

268
268
268
268
268
268
270
270
270
271
271
271
29

№ 3
0322
17

18,





1р. 17к.

ПРОГРАММА РАБОТЫ НА 2014-2015 ГОДЫ

